EPICODE

Gli oggetti: basi

Gli oggetti in JavaScript

In JavaScript praticamente ogni elemento può essere un oggetto: variabile, array, date, stringhe, espressioni matematiche.

La keyword fondamentale dell'oggetto è new. in particolare in riferimento a:

- stringhe
- numeri
- booleani

Una pratica comune suggerisce di definire un oggetto con la keyword const

```
const pet = {}
```

Per visualizzare un elemento si richiama l'oggetto con un valore:

```
pet.colore ⇒ rosso
```

Un oggetto può contenere più valori tipicamente associati ad un nome:

nome : valore

```
var pet = {
specie : "gatto",
nome : "Tigro",
colore : "rosso",
anni : 4
}
```

Gli attributi principali degli oggetti sono:

- Proprietà, cioè le caratteristiche possedute e che definiscono gli elementi dell'oggetto.
- Metodi, cioè funzioni che possono essere svolte dall'oggetto.

Le proprietà sono definite dai nomi associati ai valori

```
const pet = {
specie : "gatto",
nome : "Tigro"
}
```

specie e nome sono due proprietà dell'oggetto pet

Il metodo definito da una funzione anonima è parte integrante dell' oggetto

```
const pet = {
specie : "gatto",
nome : "Tigro",

tipo: function (){
return this.specie + this.nome;
}
}
```

this = riferimento all'oggetto corrente

Dato un certo oggetto possiamo gestire e manipolare le sue proprietà attraverso il metodo costruttore (dell'oggetto).

Analogamente alle funzioni, il costruttore accetta dei parametri per creare un modello di riferimento.

Per buona pratica il nome del costruttore è scritto in maiuscolo.

```
//Costruttore
function Pet (razza, nome, colore, anni){
this.razza = razza,
this.nome = nome,
this.colore = colore,
this.anni = anni
}

//Oggetto specifico
const gatto = new Pet("soriano", "Tigro", "rosso",
7);
```

JavaScript possiede un certo numero di oggetti nativi, quindi predefiniti.

Questi posseggono altrettanti costruttori predefiniti.

Costruttori built-in

```
new String()
new Number()
new Boolean()
new Object()
new Array()
new RegExp()
new Function()
new Date()
```

Esempio:

```
const nome = new String();
const somma = new Number
```

Ogni oggetto in JavaScript fa riferimento implicito ad un prototipo attraverso cui è possibile condividere (ereditare) proprietà e metodi.

La keyword prototype richiama questo modello di riferimento che può essere utilizzato per aggiungere proprietà e metodi ad un costruttore esistente.

```
//Costruttore
function Pet (razza, nome, colore, anni){
this.razza = razza,
this.nome = nome,
this.colore = colore,
this.anni = anni
}
```

```
//Aggiungo una proprietà al costruttore
Pet.prototype.genere = “femmina”;

//Aggiungo un metodo al costruttore
Pet.prototype.tipo = function(){
return this.razza + this.nome;
}
```

Quando abbiamo necessità di aggiungere una proprietà o un metodo ad un oggetto già esistente, possiamo farlo agevolmente:

oggetto.nuovaProprietà = valore;

oggetto.proprietà = funzione;

```
//Costruttore
function Pet (razza, nome, colore, anni){
this.razza = razza,
this.nome = nome,
this.colore = colore,
this.anni = anni
}
```

```
//Oggetto specifico
const gatto = new Pet("soriano", "Tigro", "rosso",
7);

//Aggiungo una proprietà
gatto.genere = "maschio";

//Aggiungo un metodo
gatto.tipo = function(){
return this.razza + this.nome;
}
```

EPICODE

Gli array

Array: definizione

Sostanzialmente l'array è una variabile capace di archiviare e memorizzare una serie di valori anziché uno solo alla volta.

Nel definire l'array utilizziamo la stessa keyword **var**.

Un array appare come una vera e propria lista ed è utile proprio quando i dati da inserire nel nostro programma sono una serie ordinata di valori.

Esempio una lista di animali:

```
var animale1 = "gatto";
var animale2 = "cane";
var animale3 = "coniglio";
.....
```

Array: creazione e uso

Nel definire l'array utilizziamo la stessa keyword **var**.

Indichiamo i valori che formano la lista tramite parentesi quadre e separati da virgole.

La lista di variabili:

```
var animale1 = "gatto";
var animale2 = "cane";
var animale3 = "coniglio";
```

Possiamo indicarla come array:

```
var animali = ["gatto", "cane", "coniglio"];
```

Array: creazione e uso

Possiamo indicare un array con sintassi alternative che in alcuni casi rendono il codice più leggibile.

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
var anni = [
1920,
1965,
2013
];
```

La virgola di separazione, nell'ultimo elemento non è necessaria.

Array: uso della Keyword new

Per creare un array possiamo usare la keyword new.

In questo caso la lista degli elementi è creata all'interno di parentesi toned.

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
var animali = new Array (
"gatto",
"cane",
"coniglio"
);
```

```
var anni = [
1920,
1965,
2013
];
```

```
var anni = new Array (
1920,
1965,
2013
);
```

Array come oggetti

Gli array in JS sono da considerarsi come oggetti.

L'output di typeof su un array object come tipo di dato anche la sua descrizione logica è di lista ordinata di elementi.

A questo proposito si evidenzia che in programmazione l'ordine inizia dallo 0.

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
typeof(animali);

"object"
```

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

Ordine degli elementi

gatto ⇒ 0

cane ⇒ 1

coniglio ⇒ 2

Array come oggetti

Gli array in JS sono da considerarsi come oggetti.

L'output di typeof su un array object come tipo di dato anche la sua descrizione logica è di lista ordinata di elementi.

A questo proposito si evidenzia che in programmazione l'ordine inizia dallo 0.

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

```
typeof(animali);

"object"
```

```
var animali = [
"gatto",
"cane",
"coniglio"
];
```

Ordine degli elementi

gatto ⇒ 0

cane ⇒ 1

coniglio ⇒ 2

EPICODE

Array: accesso agli elementi singoli

La possibilità di ordinare gli elementi di un array ci permette di accedere ad essi singolarmente.

Per accedere agli elementi facciamo al **numero di indice**.

```
var tipoAnimale = animali[0];
console.log(tipoAnimale); ⇒ “gatto”

var tipoAnimale = animali[1];
console.log(tipoAnimale); ⇒ “cane”

var tipoAnimale = animali[2];
console.log(tipoAnimale); ⇒ “undefined”
```

```
var animali = [
“gatto”,
“cane”,
];
```

EPICODE

Array: accesso agli elementi singoli

All'interno di un documento possiamo accedere agli elementi dell'array selezionando un elemento target nel quale visualizzare il nostro dato.

```
var animali = [
"gatto",
"cane",
];
```

```
gatto
```



```
<h1 id="test"></h1>
document.getElementById("test").innerHTML = animali[0];
```

EPICODE

Sorting di un array

Utilizzando il metodo **sort();** possiamo ordinare alfabeticamente la lista degli elementi di un array.

```
var animali = [
“gatto”,
“elefante”,
“Zebra”,
“cane”
];
```

```
> var animali = ["cane", "elefante", "gatto", "zebra"];
  animali.sort();
<· ▼(4) ["cane", "elefante", "gatto", "zebra"] i
     0: "cane"
     1: "elefante"
     2: "gatto"
     3: "zebra"
     length: 4
   ▶ __proto__: Array(0)
>
```

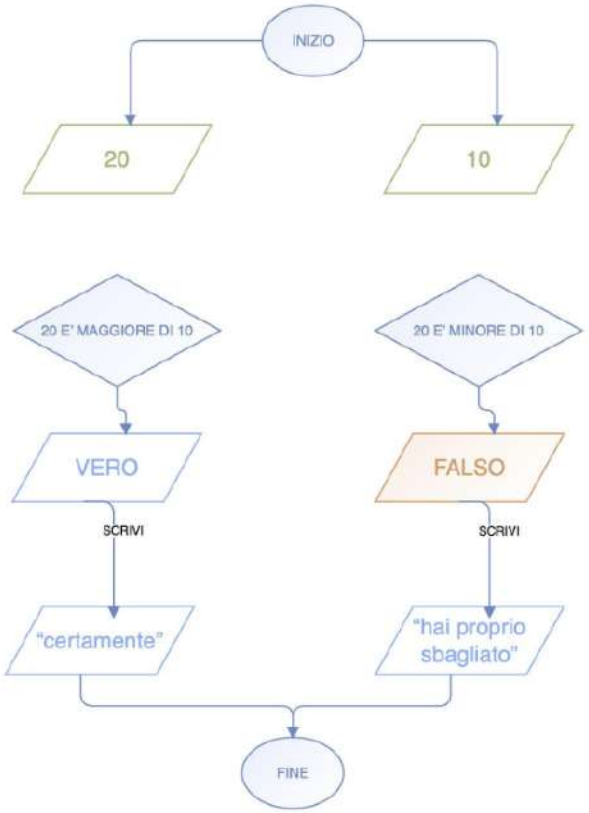
EPICODE

Introduzione alla logica condizionale e ai cicli

Metodi aritmetici random

Attraverso la logica condizionale possiamo controllare un flusso di condizioni sulla base del loro essere vere o false.

In base a questo possiamo configurare una serie di azioni differente.



La comparazione delle condizioni è una delle basi fondamentali delle strutture di logica condizionale.

Esempio, le condizioni che esprimiamo in codice, si presentano come:

Se l'utente ha meno di 18 anni oppure ha più di 16 anni, **scrivi: "può entrare"**;
Invece se ha meno di 16 anni, **scrivi: "non può entrare"**

Oppure:

Se il numero è uguale a 10 ed è minore di 15, **scrivi: "numero piccolo"**;
Invece se è uguale a 15 ed è superiore, **scrivi: "numero grande"**

Naturalmente le condizioni comparate, sulla base delle quali vengono eseguite alcune istruzioni, devono essere o vere o false.

Devono rispondere quindi, in maniera non equivoca, ad una logica booleana.

Se l'utente ha meno di 18 anni oppure ha più di 16 anni, **scrivi: "può entrare"**;

Se questa condizione sarà vera, verrà eseguita l'istruzione prescritta (scrivere il messaggio).

Altrimenti si passerà alla condizione seguente:

Invece se ha meno di 16 anni, **scrivi: "non può entrare"**

Anche la seconda condizione potrebbe essere falsa, quindi la struttura di controllo potrebbe prevedere ancora altre condizioni, fino all'avverarsi di una.

Il criterio di controllo è dato dalla comparazione di un valore tipicamente passato attraverso una variabile.

EPICODE

Il costrutto if

Il costrutto if

Il primo costrutto condizionale è indicato dalla keyword if:

```
if(condizione){
//istruzioni da eseguire;
}
```

Esprime la possibilità di eseguire una parte di codice se la condizione è vera.

var anniUtente = 13;

Se l'utente ha meno di 18 anni, scrivi: "non può entrare";

```
if(anniUtente < 18){
console.log("non puoi entrare");
}
```

I parametri del controllo di una condizione, possono combinare più comparazioni attraverso l'uso degli operatori logici.

```
> var anniUtente = 13;
  if(anniUtente < 18 && anniUtente < 16){
  "non puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 17;
  if(anniUtente < 18 && anniUtente < 16){
  "non puoi entrare";
  }
<· undefined
>
```

Avendo utilizzato l'operatore logico && and, tutte e due le condizioni devono avverarsi perchè possa essere eseguita l'istruzione.

```
> var anniUtente = 13;
  if(anniUtente < 18 || anniUtente < 16){
  "non puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 17;
  var anniUtente = 13;
  if(anniUtente < 18 || anniUtente < 16){
  "non puoi entrare";
  }
<· "non puoi entrare"
>
```

In questo caso, invece, avendo utilizzato l'operatore logico || or, una sola delle condizioni deve avverarsi perchè possa essere eseguita l'istruzione.

EPICODE

Il costrutto if/else

Con la keyword else all'interno del costrutto, possiamo definire una istruzione alternativa da eseguire se il controllo di if risulta falso.

In questo caso avendo determinato la condizione controllata da if, tutte le altre possibili condizioni, saranno controllate da else.

```
> var anniUtente = 13;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 20;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "puoi entrare"
>
```

Il costrutto if/else

Con la keyword else if all'interno del costrutto, possiamo definire una istruzione alternativa ulteriore rendendo il controllo ancora più dettagliato.

```
if( prima condizione){
//istruzioni
}else if(seconda condizione){
//istruzioni
}else{
//istruzioni
}
```

```
> var anniUtente = 13;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "non puoi entrare"
> anniUtente = 20;
  if(anniUtente < 18){
  "non puoi entrare";
  }else{
  "puoi entrare";
  }
<· "puoi entrare"
>
```