JavaScript

Objects

Francesco Marcelloni

and Alessio Vecchio

Dipartimento di Ingegneria dell'Informazione
Università di Pisa
ITALY

JavaScript Objects
Creating new objects

- Two ways for creating objects
  - use an **object initializer**
  - create **a constructor function** and then instantiate an object using that function and the **new** operator
- The first way is useful when you need to create a unique instance of the object; otherwise use the second way.
- Recent versions of JavaScript: objects can be created as instances of **classes**.

JavaScript Objects
Object Initializer

- Object initializer

  objectName = {property1:value1,... , propertyN:valueN}

  objectName: name of the new object

  propertyX: identifier (either a name, a number, or a string literal), *valueX*: expression whose value is assigned to the *propertyX*.

Object initializer: examples

```
const stud1 = {nome: 'Mario',
               cognome: 'Rossi',
               matricola: 1234};
console.log(stud1);

const o1 = {};
console.log(o1);

const o2 = {
  nome_composto: 'questo va bene',
  //altro-nome-composto: 'questo non va bene'
};
console.log(o2);
```

```
{ nome: 'Mario', cognome: 'Rossi', matricola: 1234 }
{}
{ nome_composto: 'questo va bene' }
```

Object initializer: examples

- Whe using variables, if the property name is omitted then the variable name is used:

```
const c1 = 22;
let v1 = "ABC";

const o3 = {c1, v1};
console.log(o3);
const o3bis = {c1: c1, v1: v1};
console.log(o3bis);
```

```
{ c1: 22, v1: 'ABC' }
{ c1: 22, v1: 'ABC' }
```

Accessing properties

```
// getting setting
let p1 = {x: 0, y:0};
const p2 = {x: 1, y: 2};
console.log(p1.x);
console.log(p2.y);
p1.x = 10;
p2.y = 20;
console.log(p1);
console.log(p2);

// if a property does not exist, undefined is returned
console.log(p1.z);

// A new property is added in case of write
p2.z = 33;
console.log(p2);
```

- dot notation

```
0
2
{ x: 10, y: 0 }
{ x: 1, y: 20 }
undefined
{ x: 1, y: 20, z: 33 }
```

JavaScript Objects
Object Initializer

The property of an object can be an object

The *objectName* and assignment is optional. If you do not need to refer to this object elsewhere, you do not need to assign it to a variable.

```
const cont1 = {tel: 555123456,
               email: 'mario.rossi@unipi.it'};
const s1 = {nome: "Mario",
            cognome: "Rossi",
            contatto: cont1};
console.log(s1);
function f(o) {
  console.log("Coordinates: " +
              o.x + ", " + o.y);
}
f(p1);
f({x: 100, y: 200});
```

```
{
  nome: 'Mario',
  cognome: 'Rossi',
  contatto: { tel: 555123456, email: 'mario.rossi@unipi.it' }
}
Coordinates: 10, 0
Coordinates: 100, 200
```

Objects: associative arrays notation

- There is another notation that can be used
- Properties are not limited anymore to identifier rules

```
const o4 = {
  'Una stringa e non un identificatore': 1234,
  'altra chiave': 'XYZ' };
console.log(o4);
// also mixed
const o5 = {
  'This is a string': 666,
  this_is_an_identifier: 999,
  k1: 'v1'
};
console.log(o5);
// To access:
o4['Una stringa e non un identificatore'] = 555;
o4['k1'] = 'new value';
console.log(o4);
```

```
{ 'Una stringa e non un identificatore': 1234, 'altra chiave': 'XYZ' }
{ 'This is a string': 666, this_is_an_identifier: 999, k1: 'v1' }
{
  'Una stringa e non un identificatore': 555,
  'altra chiave': 'XYZ',
  k1: 'new value'
}
```

Objects: associative arrays notation

- Now properties names can be dynamically generated

```
let a = "ABC";
let b = "DEF";
let c = Math.random();
const o6 = {
  [a+b]: "XYZ",
  [c]: Math.random()
}
console.log(o6);

const o7 = {};
for(let i=0; i<5; i++) {
  o7['k'+i]= 'v'+i;
}
console.log(o7);
```

```
{ ABCDEF: 'XYZ', '0.32245611236406035': 0.5164861542942805 }
{ k0: 'v0', k1: 'v1', k2: 'v2', k3: 'v3', k4: 'v4' }
```

for .. in

- The for...in statement loops through the enumerable properties of an object

*for (*variable *in* object*) {*

code to be executed

}

```
const studente = {nome: "Mario",
                  cognome: "Rossi",
                  matricola: 1234,
                  voti: [23, 28, 29]};
// Può anche essere const p, se non varia nel corpo del ciclo
for(let p in studente) {
  console.log(p + ' di tipo ' + typeof p);
  console.log('con valore di tipo ' + typeof studente[p]
        + ': ' + studente[p]);
}
```

nome di tipo string
con valore di tipo string: Mario
cognome di tipo string
con valore di tipo string: Rossi
matricola di tipo string
con valore di tipo number: 1234
voti di tipo string
con valore di tipo object: 23,28,29

JavaScript Objects
Object Initializer

- Object initializer (example)

```
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
let guy = { name: "Mario",
          age: 40,
          country: "Italy",
          auto: { trademark: "Ferrari", colour: "rosso" }}
</script>
</head>
<body>
<pre>
<script>
for (let a in guy)
  if (typeof(guy[a])=="object")
    for (let i in guy[a])
      document.writeln(a + '.' + i + ':' + guy[a][i]);
  else document.writeln(a + ':' + guy[a]);
</script>
</pre>
</body>
```

JavaScript Objects

- Associative array notation (example)

```
<script>
function showProperties(obj, objectName) {
  let result = "";
  // i assumes as values all the property names
  for (let i in obj)
    result += objectName + "." + i + " = " + obj[i] + "\n";
  return result;
}
</script>
</head>
<body>
<pre>
<script>
myCar = {
  make: "Fiat",
  model: "500",
  year: 2020
}
document.write(showProperties(myCar,"myCar"));
</script>
</pre>
</body>
```

JavaScript Objects
Constructor Function

- **Constructor Function**

  1. Define the object type by writing a constructor function.
  2. Create an instance of the object with **new**.

To define an object type, create a function for the object type that specifies its name, properties, and methods.

```
<script>
function Bike(make, model, year) {
  this.make = make
  this.model = model
  this.year = year
}

function displayBike(bike) {
  for (let a in bike)
    document.writeln(a + ':' + bike[a]);
  document.writeln();
}
</script>
```

```
<body>
<pre>
<script>
  mybike = new Bike("Giant", "Reign", 2013);
  mariobike = new Bike("Specialized", "Enduro", 2018);
  luigibike = new Bike("Transition", "Scout", 2020);
  displayBike(mybike);
  displayBike(mariobike);
  displayBike(luigibike);
</script>
</pre>
</body>
```

JavaScript Objects Constructor Function

```
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
function displayCar(car) {
  for (let a in car)
    if (typeof(car[a])=="object")
      for (let i in car[a])
        document.writeln(a + '.' + i + ':' + car[a][i]);
    else document.writeln(a + ':' + car[a]);
  document.writeln();
}
</script>
</head>
```

- Constructor Function

  An object can have a property that is itself another object.

```
<body>
<pre>
<script>
owner = new Person("Gino Bianchi",45,"M");
mycar = new Car("Fiat", "Topolino", 1956, owner);
luigicar = new Car("Lancia", "Stratos", 1977);
mariocar = new Car("Ford", "Model T", 1910);
displayCar(mycar);
displayCar(luigicar);
displayCar(mariocar);
</script>
</pre>
</body>
```

JavaScript Operators
new operator

- **new** operator

  You can use the new operator to create an instance of a user-defined object type or of one of the predefined object types such as **Array**, **Boolean**, **Date**, **Function**, **Image**, **Number**, **Object**, **Option**, **RegExp**, or **String**.

  objectName = new ObjectType(param1 [,param2] ...[,paramN]);

JavaScript Objects
Adding a new property

- Properties can be added to an object at run time.
- To add a property to a specific object you have to assign a value to the object:

```
car1.enginepower = 100
```

- Note: only the *car1* object will have the property *enginepower*

JavaScript Object Types
Adding a new property

- You can add a property to a previously defined object type by using the **prototype** property.
- This defines a property that is shared by all objects of the specified type, rather than by just one instance of the object.
- The following code adds a *color* property to all objects of type *Car*, and then assigns a value to the *color* property of the object *car1*:

```
Car.prototype.color = null;
car1.color="black";
```

JavaScript Object Types
Adding a new property

```
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}
function Person(name, age, sex) {
  this.name = name;
  this.age = age;
  this.sex = sex;
}
function displayCar(car) {
  for (let a in car)
    if (typeof(car[a])=="object")
      for (let i in car[a])
        document.writeln(a + '.' + i + ':' + car[a][i]);
    else document.writeln(a + ':' + car[a]);
  document.writeln();
}
</script>
</head>
```

JavaScript Object Types
Adding a new property

```
<body>
<pre>
<script>
owner = new Person("Mario Rossi",45,"M");
mycar1 = new Car("Eagle", "Talon TSi", 1993, owner);
displayCar(mycar1);

Car.prototype.color = "red";

mycar2 = new Car("Nissan", "300ZX", 1992, owner);
mycar3 = new Car("Mazda", "Miata", 1990, owner);
mycar3.color = "black";

displayCar(mycar1);
displayCar(mycar2);
displayCar(mycar3);
</script>
</pre>
</body>
```

JavaScript Objects
Defining methods

- The following syntax associates a **function** with an existing object:

  object.methodname = function_name

- You can then call the method in the context of the object as follows:

  object.methodname(params);

- You can define methods for an object type by including a method definition in the object constructor function.

JavaScript Objects
Defining methods

```
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
function Car(make, model, year, owner) {
  this.make = make
  this.model = model
  this.year = year
  this.owner = owner
  this.displayCar = displayCar;
}
function Person(name, age, sex) {
  this.name = name
  this.age = age
  this.sex = sex
}
function displayCar(){
  for (let a in this)
    if (typeof(this[a])=="object")
      for (let i in this[a])
        document.writeln(a + '.' + i + ':' + this[a][i]);
    else
      if (typeof(this[a])!="function")
        document.writeln(a + ':' + this[a]);
  document.writeln();
}
</script>
</head>
```

JavaScript Objects
Defining methods

```
<body>
<pre>
<script>
owner = new Person("Frankie Black",45,"M")
mycar1 = new Car("Eagle", "Talon TSi", 1993, owner)
mycar1.displayCar();
Car.prototype.color = "red"
mycar2 = new Car("Nissan", "300ZX", 1992,owner)
mycar3 = new Car("Mazda", "Miata", 1990,owner)
mycar3.color = "black"
mycar1.displayCar();
mycar2.displayCar();
mycar3.displayCar();
</script>
</pre>
</body>
```

JavaScript Objects
Defining methods

Alternatively:

```
function Car(make, model, year, owner) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.owner = owner;
}

Car.prototype.displayCar = function () {
  for (let a in this)
    if (typeof(this[a])=="object")
      for (let i in this[a]) document.writeln(a + '.'    + i + ':'
                                    + this[a][i]);
    else if (typeof(this[a])!="function")          document.writeln(a + ':' + this[a]);
  document.writeln();
}
```

JavaScript Operators
delete operator

delete objectName

delete objectName.property

delete objectName[index]

where objectName is the name of an object, property is an existing property, and index is an integer representing the location of an element in an array.

- If the delete operator succeeds, it sets the property or element to undefined.
- The delete operator returns *true* if the operation is possible; it returns *false* if the operation is not possible.
- You can use the delete operator to delete variables declared implicitly but not those declared with the var, let, const statement.

JavaScript Operators
delete operator

- delete operator, examples

```
x=42;
var y = 43;
myobj = new Number();
myobj.h = 4; // create property h
delete x; // returns true (can delete if declared implicitly)
delete y; // returns false (cannot delete if declared with var)
delete Math.PI; // returns false (cannot delete predefined properties)
delete myobj.h; // returns true (can delete user-defined properties)
delete myobj; // returns true (delete user-defined object)
var myobj1 = new Number();
delete myobj1; // returns false (cannot delete)
```

JavaScript Operators
delete operator

- delete operator (array elements)

  When you delete an array element, the array length is not affected.

- For example, if you delete a[3], a[4] is still a[4] and a[3] is undefined.
- When the delete operator removes an array element, that element is no longer in the array.

```
trees = new Array("redwood", "bay", "cedar", "oak", "maple");
delete trees[3];
if (3 in trees) {
   // this does not get executed
}
trees[3] = "oak";
if (3 in trees) {
   // this does get executed
}
```

Objects: methods

- Objects can include methods
- **this** for accessing the implicit object

```
const stud2 = {
  nome: 'Mario',
  cognome: 'Rossi',
  voti: [18, 21, 30],
  media() {
    let s = 0;
     for(let i=0; i<this.voti.length; i++)
       s += this.voti[i];
     return s / this.voti.length;
  },
  stampa() {
    console.log(this.nome + " " + this.cognome + " " + this.media());
  }
}
stud2.stampa();
```

```
Mario Rossi 23
```

Objects: methods

- Another notation

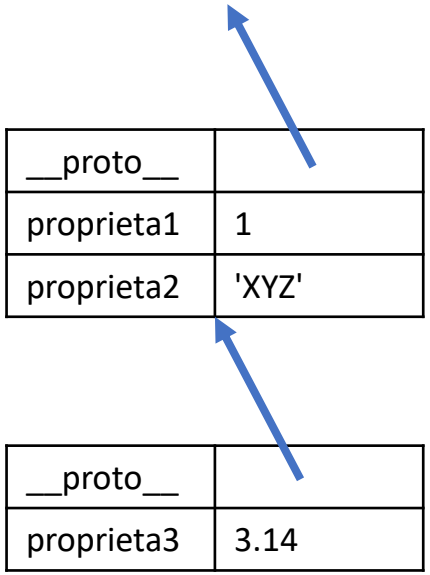
```
stud2.migliore = function () {
        let m = 0;
        for(let i=0; i<this.voti.length; i++)
                if (this.voti[i]>m)
        m = this.voti[i];
        return m;
}

console.log(stud2.migliore());
```

Inheritance

Prototype-based inheritance

- JavaScript inheritance is based on prototype objects

```
const sopra = {
  proprieta1: 1,
  proprieta2: 'XYZ'
};

const sotto = {
  __proto__: sopra,
  proprieta3: 3.14
};

console.log(sotto);
console.log(sotto.proprieta3);
console.log(sotto.proprieta1);
console.log(sotto.proprieta2);
sotto.proprieta1 = 100;
console.log(sotto);
console.log(sopra);
```




```
{ proprieta3: 3.14 }
3.14
1
XYZ
{ proprieta3: 3.14, proprieta1: 100 }
{ proprieta1: 1, proprieta2: 'XYZ' }
```

Prototype-based inheritance

- There can be a chain of prototype objects
  - sopra can have its own prototype
- Each object has its **own** properties and **inherited** properties
  - own properties of sopra: proprieta1 and proprieta2
  - own properties of sotto: proprieta3
  - inherited properties of sotto: proprieta1 and proprieta2
  - some methods consider just own properties, other consider also the inherited ones
- When reading the property of an object
  - if the object does not have such a property the prototype chain is followed looking for such a property
  - if not found: undefined
- When writing a value in an inherited property, the object is given such a property with the new value
  - the new property hides the corresponding one in the prototype object
  - the value of the property in the prototype remains unchanged

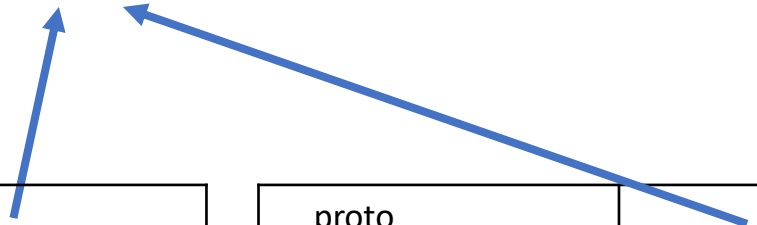
Prototype-based inheritance

```
const Persona = {
  stampaGeneralita() {
    console.log(this.nome + " " + this.cognome);
  }
};
const studenteA = {
  __proto__: Persona,
  nome: "Sara",
  cognome: "Verdi",
  media: 23.5,
  stampaStudente() {
    this.stampaGeneralita();
    console.log("media: " + this.media);
  }
}
const docenteX = {
  __proto__: Persona,
  nome: "Paolo",
  cognome: "Bianchi",
  email: "paolo.bianchi@unipi.it",
  stampaDocente() {
    this.stampaGeneralita();
    console.log("email: " + this.email);
  }
}
studenteA.stampaStudente();
docenteX.stampaDocente();
```

- It works also for methods

Sara Verdi
media: 23.5
Paolo Bianchi
email: paolo.bianchi@unipi.it

__proto__	...
stampaGeneralita	function () {...}



__proto__	
nome	Paolo
cognome	Bianchi
email	paolo.bianchi@unipi.it
stampaDocente	function () {...}

__proto__	
nome	Sara
cognome	Verdi
media	23.5
stampaStudente	function () {...}

Constructor functions

- The __*proto*__ property is generally not set by hand
- Constructor functions allows the programmer to create instances of classes using the new operator
- The function automatically set the __*proto*__ property for all created object so that they all inherit from a common prototype object
- The prototype object contains the method inherited by all class instaces
- In JavaScript, two objects are instances of the same class if they have the same prototype object

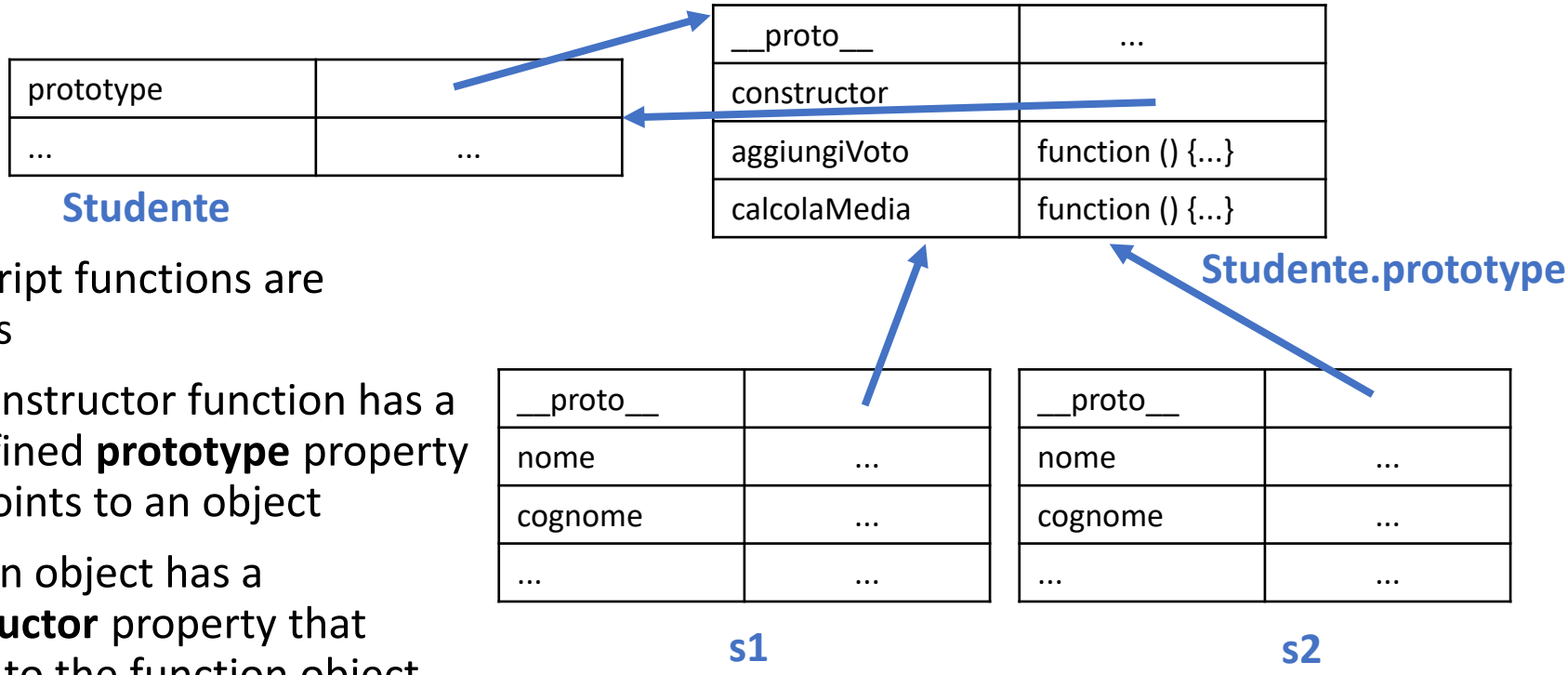
Constructor functions

```
function Studente(n, c, m) {
  this.nome = n;
  this.cognome = c;
  this.matricola = m;
  this.voti = [];
}

Studente.prototype.aggiungiVoto = function(v) {
  this.voti[this.voti.length] = v;
}

Studente.prototype.calcolaMedia = function() {
  let s = 0;
  for(let i=0; i<this.voti.length; i++)
    s += this.voti[i];
  return s / this.voti.length;
}
```

Constructor functions

```
let s1 = new Studente("Mario", "Rossi", 1234);
s1.aggiungiVoto(20);
s1.aggiungiVoto(22);
let media = s1.calcolaMedia();
console.log(media);
const s2 = new Studente("Sara", "Verdi", 8899);
s2.aggiungiVoto(30);
console.log(s2.calcolaMedia());
```

1. creates a new object and sets its **__proto__** property to the **Studente.prototype** object
2. the body of the **Studente()** function is executed, **this** is the newly created object
3. the **Studente.prototype** object has been equipped the **aggiungiVoto()** and **aggiungiMedia()** methods, that are hence inherited



- JavaScript functions are objects
- The constructor function has a predefined **prototype** property that points to an object
- Such an object has a **constructor** property that points to the function object
- The **instanceof** operator:

  s instanceof Studente

  returns true if **Studente.prototype** is in the inheritance chain for object **s**

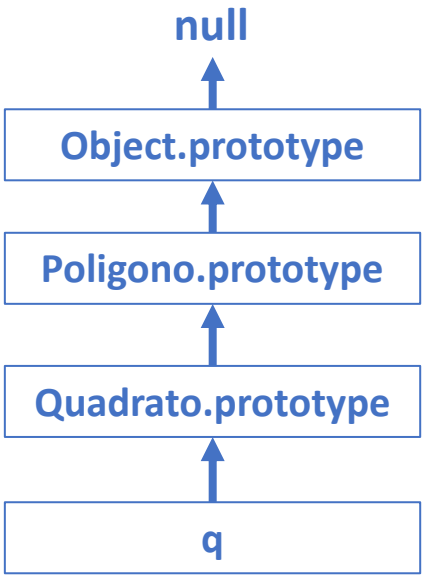
```
console.log(s1.__proto__ === s2.__proto__);
// true
console.log(s1.__proto__ === Studente.prototype);
// true
console.log(Studente.prototype.constructor === Studente);
// true
```

Setting/getting prototype

- __proto__ is also indicated as [[Prototype]]
  - more precisely: __proto__ is the accessor of [[Prototype]]
- Direct manipulation should be avoided, use instead
  - *Object.getPrototypeOf(x)* to obtain the *__proto__* of *x*
  - *Object.setPrototypeOf(x, p)* sets the *__proto__* of *x* to *p*
- The *Object.create(p)* method creates a new object that inherits from *p*

```
const myp = {a: 1, b: 2};
const mysub = Object.create(myp);
console.log(mysub.a);
// Prints 1
const newp = {c: 3};
Object.setPrototypeOf(mysub, newp);
console.log(mysub.a);
// Prints undefined
console.log(mysub.c);
// Prints 3
```

Inheritance




```
function Poligono(n, l) {
  this.lunghezzaLato = l;
  this.numeroLati = n;
}

function Quadrato (l) {
  Poligono.call(this, 4, l);
}

Object.setPrototypeOf(Quadrato.prototype,
                      Poligono.prototype);

Poligono.prototype.perimetro = function () {
  return this.lunghezzaLato * this.numeroLati;
}

Quadrato.prototype.area = function () {
  return this.lunghezzaLato *
        this.lunghezzaLato;
}

const q = new Quadrato(10);
console.log(q.area());                  // 100
console.log(q.perimetro());             // 40
```

Built-in Objects

Objects

- Predefined built-in objects

  Array, Boolean, Date, Function, Math, Number, RegExp, and String

- Predefined client-side objects
  - When you load a document in the browser, it creates a number of JavaScript objects with property values based on the HTML in the document and other pertinent information.
  - These objects exist in a hierarchy that reflects the structure of the HTML page itself.
  - Each page has always the objects: Navigator, Window, Document, Location, History.

Built-in Objects
Array

- An array is an ordered set of values that you refer to with an index
- Array elements may belong to **different** types
- The Array object has methods for manipulating arrays in various ways, such as joining, reversing, and sorting them
- Methods are inherited (*Array.prototype*)
- Each array has a property for determining the array length

Built-in Objects
Array

- Create an Array

  Three forms

  - let myCars = new Array(); // (add an optional integer
    myCars[0] = "Saab"; // argument to control array's size)
    myCars[1] = "Volvo";
    myCars[2] = "BMW";

  - let myCars = new Array("Saab", "Volvo", "BMW"); //condensed

  - let myCars = ["Saab", "Volvo", "BMW"]; //literal array
    - It is also possible to avoid to specify all the values of the elements of the array

    let myCars = ["Saab", ,"BMW"];

Built-in Objects
Array

- Access an Array

```
document.write(myCars[0]);
```

- Modify values in an Array

```
myCars[0] = "Ferrari";
```

- An array can dynamically be extended

```
myCars[6] = "Fiat";
```

The size of the array is extended to 7.

The elements with indexes 3, 4 and 5 are undefined.

Built-in Objects
Array

- The length property specifies the size of the array

```
let v = [10, 20, 30];
console.log(v.length);
v.length = 6;
console.log(v);
v[8] = 999;
console.log(v);
```

```
3
[ 10, 20, 30, <3 empty items> ]
[ 10, 20, 30, <5 empty items>, 999 ]
```

Built-in Objects
Array

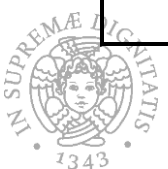
Method	Description
concat()	Joins two or more arrays, and returns a copy of the joined arrays
	array.concat(array2, array3, ..., arrayX)
join()	Joins all elements of an array into a string
	array.join(separator)
	If the separator is omitted, the elements are separated with a comma
pop()	Removes the last element of an array, and returns that element
	array.pop()
push()	Adds new elements to the end of an array, and returns the new length
	array.push(element1, element2, ..., elementX)

Built-in Objects
Array

Method	Description
reverse()	Reverses the order of the elements in an array
	array.reverse()
shift()	Removes the first element of an array, and returns that element
	array.shift()
slice()	Selects a part of an array, and returns the new array
	array.slice(start, end)
	start Required. An integer that specifies where to start the selection. You can also use negative numbers to select from the end of an array
	end Optional. An integer that specifies where to end the selection. If omitted, slice() selects all elements from the start position and to the end of the array

Built-in Objects
Array

Method	Description
sort()	Sorts the elements of an array
	array.sort(sortfunc)
	sortfunc Optional. A function that defines the sort order Default: sorts the elements alphabetically and ascending. However, numbers will not be sorted correctly (40 comes before 5). To sort numbers, you must add a function that compare numbers
splice()	Adds/Removes elements from an array
	array.splice(index,howmany,element1,.....,elementX)
	index Required. An integer that specifies at what position to add/remove elements
	howmany Required. The number of elements to be removed. If set to 0, no elements will be removed
	element1, ..., elementX Optional. The new element(s) to be added to the array



Built-in Objects
Array

Method	Description
toString()	Converts an array to a string, and returns the result
	array.toString()
unshift()	Adds new elements to the beginning of an array, and returns the new length
	array.unshift(element1,element2, ..., elementX)
	element1,element2, ..., elementX Required. The element(s) to add to the beginning of the array
valueOf()	Returns the primitive values of an array
	array.valueOf()

Built-in Objects
Array

- Sort numbers

```
<script>
function compare(a, b) {
        return a - b;       //(numerically and ascending)
       //return b - a;   (numerically and descending)
}
let n = [10, 5, 40, 25, 100, 1];
document.write(n.sort(compare));
</script>
```

If compare(a, b) < 0, a is placed before b;
If compare(a, b) == 0, no change;
compare(a, b) > 0, b is placed before a;

Built-in Objects
Array

- Example

```
<head>
<meta charset="utf-8">
<title>Arrays</title>
<script>
let myvector= new Array("S","D","A","B");
print(myvector,"(initial)");

function add() {
  myvector[myvector.length] = document.forms[0].text1.value;
  print(myvector, "(add)");
}
function concatenate() {
  let vector = myvector.join("+");
  document.writeln(vector+ "<br>(concatenate)");
}
function invert() {
  myvector.reverse();
  print(myvector, "(invert)");
}
```

Built-in Objects
Array

```
function shift() {
  myvector.shift();
  print(myvector,"(shift)");
}

function sort() {
  myvector.sort();
  print(myvector,"(sort)");
}

function print(vector, comment) {
  for (let i = 0; i < vector.length; i++)
    document.writeln(vector[i]);
  document.writeln("<br>"+ comment);
}
</script>
</head>
```

Built-in Objects
Array

```
body>
<form action="#">
<p>Add a new element: <input type="text" name="text1">
<input type="button" value="ADD" onclick="add()"><br>
<input type="button" value="JOIN('+')" onclick="concatenate()"><br>
<input type="button" value="INVERT" onclick="invert()"><br>
<input type="button" value="SHIFT" onclick="shift()"><br>
<input type="button" value="SORT" onclick="sort()"><br>
</p>
</form>
</body>
```

Built-in Objects
Boolean

Used in two ways:

- **new Boolean(v)** creates a Boolean wrapper object, it is not a primitive value
- **Boolean(v)** convert v to a boolean primitive value

The second form much more common than the first one

new Boolean()

```
// Some wrapper objects
const b1 = new Boolean(0);
const b2 = new Boolean(null);
const b3 = new Boolean(false);
const b4 = new Boolean("ABC");
const b5 = new Boolean(39);

// valueOf() retrieves the primitive
// value (true/false) from the wrapper object
console.log(b1.valueOf());
console.log(b2.valueOf());
console.log(b3.valueOf());
console.log(b4.valueOf());
console.log(b5.valueOf());

// The type of b1, ..., b5 is object
console.log(typeof b1);

// A non null reference is considered true, also
// when it is a Boolean contaning a false value
if(b3) {
  console.log("I'm here: then");
} else {
  console.log("I'm here: else");
}
// Now the primitive value
if(b3.valueOf()) {
  console.log("I'm here: then");
} else {
  console.log("I'm here: else");
}
```

```
false
false
false
true
true
object
I'm here: then
I'm here: else
```

Boolean

- undefined is converted to false
- null is converted to false
- 0, -0, and NaN are converted to false; other numbers converted to true
- All objects converted to true
- Empty string converted to false

```
const cb1 = Boolean(undefined);
const cb2 = Boolean(null);
const cb3 = Boolean(0);
const cb4 = Boolean([]);
const cb5 = Boolean('');
const cb6 = Boolean({a: 1});
const cb7 = Boolean([1, 2, 3]);
const cb8 = Boolean('ABC');
const cb9 = Boolean(
        new Boolean(false));
console.log(cb1);
...
console.log(cb9);
```

```
false
false
false
true
false
true
true
true
true
```

Boolean

- Values that are converted to true are said *truthy*
- Values that are converted to false are said *falsy*
- Things can be weird...

```
// Empty array is truthy
let a = [];
if(a) {
  // all objects are truthy and an empty array is still an object
  console.log("I'm here: then");
} else {
  console.log("I'm here: else");
}
// but it is loosely equal to false
if(a == false) {
  // Multiple conversions
  // a is converted to a primitive using Array.prototype.toString()
  // the result is ""
  // When converting string and boolean both are converted to numbers
  // and they are both 0
  console.log("I'm here: then");
} else {
  console.log("I'm here: else");
}
```

```
I'm here: then
I'm here: then
```

Built-in Objects
Date

- JavaScript stores dates as the number of milliseconds since January 1, 1970, 00:00:00
- Four forms to create a Date object
  - new Date () *// Date object with current date and time*
  - new Date (milliseconds) *//milliseconds since 1970/01/01*
  - new Date (dateString) *//A string representing a date in the following form: "Month day, year hours:minutes:seconds." For example, Xmas95 = new Date("December 25, 1995 13:30:00"). If you omit hours, minutes, or seconds, the value will be set to zero.*
  - new Date (year, month, day, hours, minutes, seconds, milliseconds) *//parameters are integer values*
- Most parameters above are optional. Not specifying them, causes 0 to be passed in.

Built-in Objects
Date

Method	Description
getDate()	Returns the day of the month (from 1-31)
getDay()	Returns the day of the week (from 0-6)
getFullYear()	Returns the year (four digits)
getHours()	Returns the hour (from 0-23)
getMilliseconds()	Returns the milliseconds (from 0-999)
getMinutes()	Returns the minutes (from 0-59)
getMonth()	Returns the month (from 0-11)
getSeconds()	Returns the seconds (from 0-59)
getTime()	Returns the number of milliseconds since midnight Jan 1, 1970

Built-in Objects
Date

Method	Description
getUTCDate()	Returns the day of the month, according to universal time (from 1-31). UTC (Coordinated Universal Time)
getUTCDay()	Returns the day of the week, according to universal time (from 0-6)
getUTCFullYear()	Returns the year, according to universal time (four digits)
getUTCHours()	Returns the hour, according to universal time (from 0-23)
getUTCMilliseconds()	Returns the milliseconds, according to universal time (from 0-999)
getUTCMinutes()	Returns the minutes, according to universal time (from 0-59)
getUTCMonth()	Returns the month, according to universal time (from 0-11)
getUTCSeconds()	Returns the seconds, according to universal time (from 0-59)

Built-in Objects
Date

Method	Description
parse()	Parses a date string and returns the number of milliseconds since midnight of January 1, 1970
	Date.parse(datestring)
	Datestring Required. A string representing a date
setDate()	Sets the day of the month (from 1-31)
setFullYear()	Sets the year (four digits)
setHours()	Sets the hour (from 0-23)
setMilliseconds()	Sets the milliseconds (from 0-999)
setMinutes()	Set the minutes (from 0-59)
setMonth()	Sets the month (from 0-11)
setSeconds()	Sets the seconds (from 0-59)
setTime()	Sets a date and time by adding or subtracting a specified number of milliseconds to/from midnight January 1, 1970

Built-in Objects
Date

Method	Description
setUTCDate()	Sets the day of the month, according to universal time (from 1-31)
setUTCFullYear()	Sets the year, according to universal time (four digits)
setUTCHours()	Sets the hour, according to universal time (from 0-23)
setUTCMilliseconds()	Sets the milliseconds, according to universal time (from 0-999)
setUTCMinutes()	Set the minutes, according to universal time (from 0-59)
setUTCMonth()	Sets the month, according to universal time (from 0-11)
setUTCSeconds()	Set the seconds, according to universal time (from 0-59)

Built-in Objects
Date

Method	Description
toDateString()	Converts the date portion of a Date object into a readable string
toLocaleDateString()	Returns the date portion of a Date object as a string, using locale conventions
toLocaleTimeString()	Returns the time portion of a Date object as a string, using locale conventions
toLocaleString()	Converts a Date object to a string, using locale conventions
toString()	Converts a Date object to a string
toTimeString()	Converts the time portion of a Date object to a string
toUTCString()	Converts a Date object to a string, according to universal time
UTC()	Returns the number of milliseconds in a date string since midnight of January 1, 1970, according to universal time
valueOf()	Returns the primitive value of a Date object

Built-in Objects
Date

```
<script type="text/javascript">
        var d = new Date();
        document.write("Original form: ");
        document.write(d + "<br>");
        document.write("Formatted form: ");
        document.write(d.toLocaleDateString());
</script>
```

Output

Original form: Sun Oct 24 2021 23:18:24 GMT+0200 (Ora legale dell'Europa centrale)

Formatted form: 24/10/2021

Built-in Objects
Math

- The Math object allows performing mathematical tasks.
- The Math object includes several mathematical constants and methods.

- Math is not a constructor. All properties/methods of Math can be called by using Math as an object, without creating it.
- Note that all trigonometric methods of Math take arguments in radians.

```
let pi_value = Math.PI;
let sqrt_value = Math.sqrt(16);
```

Built-in Objects
Math

- Note: Case-Sensitive

Property	Description
E	Returns Euler's number (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.442)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14159)
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)

Built-in Objects
Math

Method	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x, in radians
asin(x)	Returns the arcsine of x, in radians
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
atan2(y,x)	Returns the arctangent of the quotient of its arguments
ceil(x)	Returns x, rounded upwards to the nearest integer
cos(x)	Returns the cosine of x (x is in radians)
exp(x)	Returns the value of E^x

Built-in Objects
Math

Method	Description
floor(x)	Returns x, rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of x
max(x,y,z,...,n)	Returns the number with the highest value
min(x,y,z,...,n)	Returns the number with the lowest value
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds x to the nearest integer
sin(x)	Returns the sine of x (x is in radians)
sqrt(x)	Returns the square root of x
tan(x)	Returns the tangent of an angle

Built-in Objects
Number

Used in two ways:

- **new Number(v)** creates a Number wrapper object, it is not a primitive value
- **Number(v)** convert v to a number primitive value

The second form much more common than the first one

Built-in Objects
Number

- The Number object has properties for numerical constants, such as maximum value, not-a-number, and infinity.
- You cannot change the values of these properties and you use them as follows:

  biggestNum = Number.MAX_VALUE

  smallestNum = Number.MIN_VALUE

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
POSITIVE_INFINITY	Represents infinity (returned on overflow)

Built-in Objects
Number

Method	Description
toExponential(x)	Converts a number into an exponential notation
	number.toExponential(x)
	x Optional. An integer between 0 and 20 representing the number of digits in the notation after the decimal point. If omitted, it is set to as many digits as necessary to represent the value
toFixed(x)	Formats a number with x numbers of digits after the decimal point
	number.toFixed(x)
	x Optional. The number of digits after the decimal point. Default is 0 (no digits after the decimal point)
toPrecision(x)	Formats a number to x length
	number.toPrecision(x)
	x Optional. The number of digits. If omitted, it returns the entire number (without any formatting)
toString()	Converts a Number object to a string
valueOf()	Returns the primitive value of a Number object

Built-in Objects
Number

```
<script>
let num = new Number(13.3714);
document.write(Number.MAX_VALUE+"<br>");
document.write(Number.MIN_VALUE+"<br>");
document.write(num.toExponential(4)+"<br>");
document.write(num.toFixed(2)+"<br>");
document.write(num.toPrecision(3)+"<br>");
document.write(num.toString()+"<br>");
document.write(num.valueOf());
</script>
```

Number

- Number(value), used as a function, converts a value to a number. If the value cannot be converted, it returns NaN
  - null is converted to 0
  - undefined is converted to NaN
  - true and false are converted to 1 and 0
  - string are parsed, in case of failure NaN
  - objects are converted to primitive values
- Number.parseInt(string)/ Number.parseFloat(string): similar, but just for strings

```
const n1 = Number('ABC');
const n2 = Number('123');
const n3 = Number('');
const n4 = Number([]);
const n5 = Number([33]);
const n6 = Number([33, 44]);
const n7 = Number(undefined);
const n8 = Number(null);
const n9 = Number(true);
const n10 = Number(false);
const n11 = Number({valore: 4,
                valueOf() {
          return this.valore;
         }});
```

```
NaN
123
0
0
33
NaN
NaN
0
1
0
4
```

Built-in Objects
String

- The String object is a wrapper around the string primitive data type. Do not confuse a string literal with the String object.
- String wrapper objects are created with new String().

```
s1 = "Hi"                 //creates a string literal value
s2 = new String("Hi")     //creates a String object
```

- Similarly to Boolean and Number, String is more frequently used without new, as a conversion function. But mainly, it is a container for methods.

String

- Conversions:
  - undefined is converted to "undefined"
  - null is converted to "null"
  - true and false are converted to "true" and "false"
  - numbers are converted to base 10
  - objects are converted to a primitive (toString() and then valueOf())

```
const o = {valore: 4,
  toString() {
    return "pippo"
  },
  valueOf() {
    return this.valore;
  }};
const s1 = String(undefined);
const s2 = String(null);
const s3 = String(1234);
const s4 = String(true);
const s5 = String(o);
const s6 = "" + o;
```

```
undefined
null
1234
true
pippo
4
```

"" + o should not be used for string coercion as the primitive value of o is obtained differently (priority to valueOf)

Built-in Objects
String

- You can call any of the methods of the String object on a string literal value
  - JavaScript automatically converts the string literal to a temporary String object, calls the method, then discards the temporary String object.
- You can also use the String.length property with a string literal.

Built-in Objects
String

Method	Description
charAt()	Returns the character at the specified index
charCodeAt()	Returns the Unicode of the character at the specified index
concat()	Joins two or more strings, and returns a copy of the joined strings
	string.concat(string2, string3, ..., stringX)
fromCharCode()	Converts Unicode values to characters
indexOf()	Returns the position of the first found occurrence of a specified value in a string
lastIndexOf()	Returns the position of the last found occurrence of a specified value in a string (-1 if the value to search for never occurs)
	string.lastIndexOf(searchstring, start)
	searchstring Required. The string to search for **Start** Optional. The start position in the string to start the search. If omitted, the search starts from position 0

Built-in Objects
String

Method	Description
match()	Searches for a match between a regular expression and a string, and returns the matches
	string.match(regexp)
	regexp Required. A regular expression.
replace()	Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring
	string.replace(*regexp/substr,newstring*)
	regexp/substr Required. A substring or a regular expression. *newstring* Required. The string to replace the found value in parameter 1
search()	Searches for a match between a regular expression and a string, and returns the position of the match

Built-in Objects
String

Method	Description
slice()	Extracts a part of a string and returns a new string *string*.slice(begin,end) *begin* Required. The index where to begin the extraction. First character is at index 0 *end* Optional. Where to end the extraction. If omitted, slice() selects all characters from the begin position to the end of the string
split()	Splits a string into an array of substrings *string*.split(separator, limit) *separator* Optional. Specifies the character to use for splitting the string. If omitted, the entire string will be returned *limit* Optional. An integer that specifies the number of splits

Built-in Objects
String

Method	Description
substr()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the characters from a string, between two specified indices
toLowerCase()	Converts a string to lowercase letters
toUpperCase()	Converts a string to uppercase letters
valueOf()	Returns the primitive value of a String object

Built-in Objects
String

```
<script>
let str="Hello world!";
document.write(str +"<br>");
document.write(str.substring(1) +"<br>");
document.write(str.substring(3,7)+"<br>");
document.write(str.split('o',2)+"<br>");
document.write(str.toUpperCase()+"<br>");
document.write(str.toLowerCase());
</script>
```

JavaScript
RegExp

- A regular expression is an object that describes a pattern of characters.
  - A simple pattern can be one single character.
  - A more complicated pattern can consist of more characters, and can be used for parsing, format checking, substitution and more.

- Regular expressions are used to perform powerful pattern-matching and "search-and-replace" functions on text.

```
var txt = new RegExp(pattern, modifiers);
or more simply
var txt = /pattern/modifiers;
```

pattern specifies the pattern of an expression

modifiers specify if a search should be global, case-sensitive, etc.

Built-in Objects
RegExp

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Method	Description
exec()	Tests for a match in a string. Returns the first match regexp.exec(str) regexp(str)
test()	Tests for a match in a string. Returns true or false regexp.test(str)

Built-in Objects
RegExp

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character not between the brackets
[0-9]	Find any digit from 0 to 9
[a-z]	Find any character from lowercase a to lowercase z
[A-Z]	Find any character from uppercase A to uppercase Z
[a-Z]	Find any character from lowercase a to uppercase Z
adgk	Find the sequence of characters
(red|blue|green)	Find any of the alternatives specified

Built-in Objects
RegExp

Metachar	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word

Built-in Objects
RegExp

Metachar	Description
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

Built-in Objects
RegExp

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{X}	Matches any string that contains a sequence of X n's
n{X,Y}	Matches any string that contains a sequence of X or Y n's
n{X,}	Matches any string that contains a sequence of at least X n's
n$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it
?=n	Matches any string that is followed by a specific string n
?!n	Matches any string that is not followed by a specific string n

Built-in Objects
RegExp

```
<body>
<script>
let s1 = "10, 200, 2000, 10000 or 300000?";
let p1 = /\d{3,4}/g;
let v1 = s1.match(p1);
document.write(v1);
document.write("<br>");
let s2 = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, " +
  "sed do eiusmod tempor incididunt ut labore et dolore a magna aliqua.";
let p2 = /[abc]\w+/g;
let v2 = s2.match(p2);
document.write(v2);
</script>
</body>
```

Built-in Objects
Global Properties and Functions

- The JavaScript global properties and functions can be used with all the built-in JavaScript objects.

Property	Description
Infinity	A numeric value that represents positive/negative infinity
NaN	"Not-a-Number" value
undefined	Indicates that a variable has not been assigned a value

Function	Description
decodeURI()	Decodes a URI
decodeURIComponent()	Decodes a URI component
encodeURI()	Encodes a URI
encodeURIComponent()	Encodes a URI component

Built-in Objects
Global Properties and Functions

Function	Description
escape()	Encodes a string
eval()	Evaluates a string and executes it as if it was script code. First, eval() determines if the argument is a valid string, then eval() parses the string looking for JavaScript code. If it finds any JavaScript code, it will be executed.
isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string
unescape()	Decodes an encoded string

Built-in Objects
Global Properties and Functions

```
<script type="text/javascript">
        eval("x=10;y=20;document.write(x*y)");
        document.write("<br>" + eval("2+2"));
        document.write("<br>" + eval(x+17));
</script>
```

Output:

200

4

27