JavaScript

Control structure, function

Data type conversion

- JavaScript automatically converts values when it assigns values to a variable or evaluates an expression.
- Most of the time, letting JavaScript handle the data works fine, but there are times when you want to force a conversion of one type to another.
- For example, if you prompt a user for input, the input is set as a string. But, suppose you want to perform calculations on the incoming data, making it necessary to convert the strings to numbers.

Primitive data type functions

JavaScript provides three functions to convert the primitive data types. They are:

- *String()*
- *Number()*
- *Boolean()*

Example

```
<html>
<head><title>The Conversion Functions</title></head>
<body>
<script type="text/javascript">
var num1 = prompt("Enter a number: ","");    //20
var num2 = prompt("Enter another number: ","");         //30

var result = Number(num1) + Number(num2);  // strings to numbers
    alert("Result is "+ result);                        //50

var myString=String(num1);
result=myString + 200;                     // String + Number = String
    alert("Result is "+ result);           //20200

    alert("Boolean result is "+ Boolean(num2));  // Prints true
</script>
</body>
</html>
```

Convert to Int

- ***parseInt()***
  - parseInt(String, NumberBase); *Default base is 10*
  - parseInt(String);
  - parseInt("111", 2); 7 *(111 in base 2 is 7)*
  - parseInt("45days"); *45*

String	***Result***
"hello"	*NaN*
"Route 66"	*NaN*
"6 dogs"	*6*
"6"	*6*
"-6"	*–6*
"6.56"	*6*
"0Xa"	*10*
"011"	*9*

Convert to Int on the Base

String	**Base**	**Result (Decimal)**
"111"	*2 (binary)*	*7*
"12"	*8 (octal)*	*10*
"b"	*16 (hex)*	*11*

Convert to float

- ***parseFloat()***

String	***Result***
"hello"	*NaN*
"Route 66.6"	*NaN*
"6.5 dogs"	*6.5*
"6“	*6*
"6.56“	*6.56*

String evaluation

- ***eval()***

  eval(String);

- var result= eval("(5+4) / 3");
- ***alert(result)   // displays 3***

Control Structures, Blocks, and Compound Statements

Conditionals

if/else	***basic comparision***
?:	***shorcut of else/if***
if/else if	***multiway decision structure***
switch	***more readable on multiple options***

```
switch (color){
case "red":  alert("Hot!");                           break;
case "blue“:         alert("Cold.");                          break;
default:      alert("Not a good choice.");          break;
}
```

Loops

- ***while***
- ***do while***
- ***for***
- ***for in***
- ***for of***

control statements

- ***break*** *Exits the loop to the next statement after the closing curly brace of the* loop's statement block.
- **continue** *Sends loop control directly to the top of the loop and re-evaluates the loop* condition. If the condition is true, enters the loop block.

continue vs break

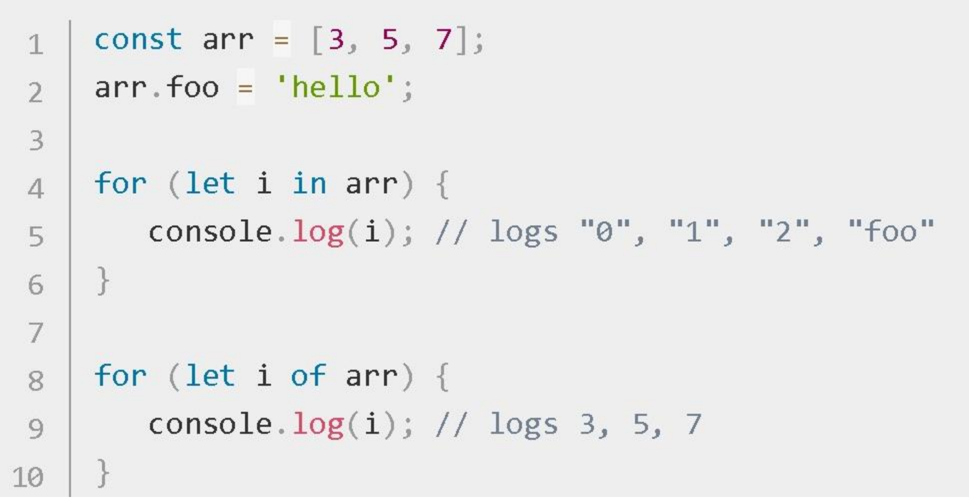
```
while(true) {
var grade=eval(prompt("What was your grade? ",""));
if (grade < 0 || grade > 100)
{   alert("Illegal choice!");
    continue; // Go back to the top of the loop }

if(grade > 89 && grade < 101)
{   alert("Wow! You got an A!");}
    else if (grade > 79 && grade < 90)
    {alert("You got a B");}
            else if (grade > 69 && grade < 80)
            {alert("You got a C");}
                        else if (grade > 59 && grade < 70)
                        {alert("You got a D");}
                        else {alert("Study harder.You Failed.");}

                        answer=prompt("Do you want to enter another grade?","");
                        if(answer != "yes"){
                        break; // Break out of the loop to line 12
}
}
```

Label

```
<html>
<body>
<script type="text/javascript">
outerLoop:    for ( var row = 0; row < 10; row++){
                        for ( var col=0; col <= row; col++){
                        document.write("row "+ row +"|column " + col, "<br />");
                                    if(col==3){
                                    document.write("Breaking out of outer loop  at " + col +"<br />");
                        break outerLoop;
}}
document.write("*************<br />");
} // end outer loop block
</script>
</body>
</html>
```

for .. in VS for .. of

- The for...in statement iterates a specified variable over all the enumerable properties of an object.

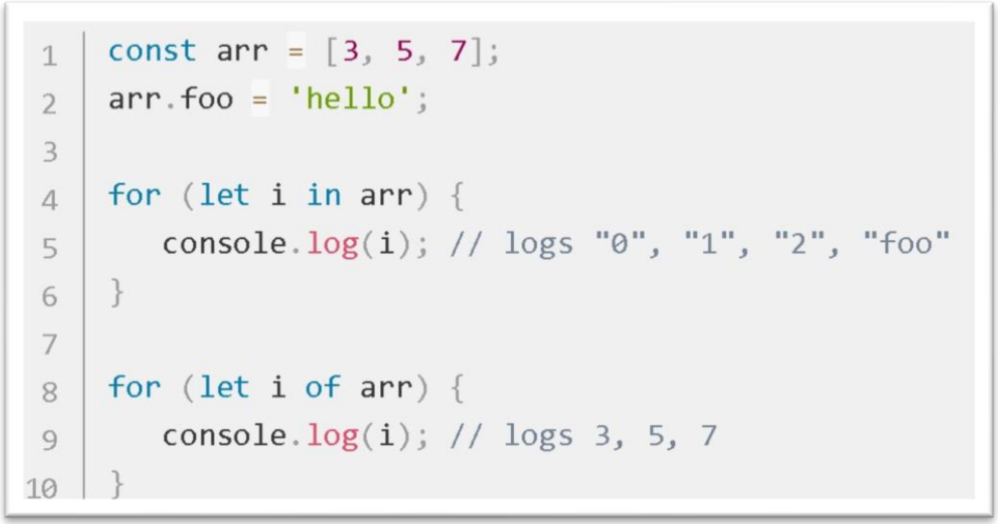


```
const arr = [3, 5, 7];
arr.foo = 'hello';

for (let i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
   console.log(i); // logs 3, 5, 7
}
```

for loop

- The *for...in* statement iterates a specified variable over all the enumerable properties of an object.

- The *for...of* statement creates a loop Iterating over iterable objects (including Array, Map, Set, arguments object and so on)



```
const arr = [3, 5, 7];
arr.foo = 'hello';

for (let i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
   console.log(i); // logs 3, 5, 7
}
```

Functions

Function Declaration and Invocation

Declaration

- function *function_name (parameter){ statement;* }

Invocation /Function call/

- Calling from the code
  - **function_name(“hello”);**

- Calling from a Link.
  - **<a href=“javascript:function_name()“></a>**

- Calling from an Event
  - **<input type="button” value=“Hi” onClick=“function_name('Dan');“ />**

Function usage example

```
<html>

<head><title>A Simple Function</title>
<script type="text/javascript">
function SayHello(myparam){ // Function defined within <head> tags
alert( myparam )}
</script>
</head>

<body >
<script type="text/javascript">
SayHello(“Hello”);
</script>
</body></html>
```

Scope of Variables in Functions

```
<html>
<head><title>Function Scope</title>
<script type="text/javascript">
var name=“Bat"; // Global variable
var hometown=“UB";

function greetme(){
    var name=“Dorj"; // Local variable
    var hometown=“UM";
    alert("In function the name is " + name +" and hometown is "+ hometown);
}
</script>
</head>
<body><script type="text/javascript">
greetme(); // Function call
alert("Out of function, name is "+ name +” and hometown is " + hometown);
</script>
</body>
</html>
```

Return value

```
function sum (a, b) {
var result= a + b;
return result;
}

var total=sum(5, 10);
```

Anonymous Functions as Variables

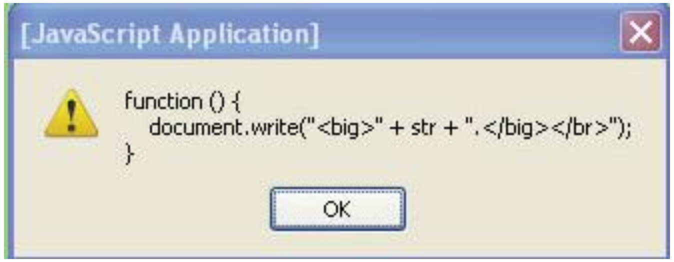
```
window.onload = function() {
                                                    alert("Welcome");}
```

Function body is assigned to greetings

```
<head>…
var greetings = function (visitor)
{
message="Greetings to you, " + visitor + "! ";
return message;
}

<body>…

text=greetings;      // greetings is a variable, its value is the function definition
document.write(text +"<br />");
text=greetings();    // Call function
document.write(text +"<br />");
```

```
// Function body is assigned to greetings
var greetings=function (visitor){
message="Greetings to you, " + visitor + "! ";
return message;
}

var salutation = greetings("Elfie");
document.write(salutation + "<br />");
var hello = greetings;
// Function variable assigned to another variable
var welcome = greetings;
document.write( hello("Stranger") +"<br />“ );
// Call the function
document.write( welcome ( "your Majesty“ ) +" May I take your coat? </br />“ );
```

```
function paint(type, color) {
var str = "The " + type + " is " + color;        //local variable
var tellme = function() {                        // Anonymous function
                   document.write("<big>"+str+".</big><br />")
                   }
return tellme;             // return a reference to the function
}


…
// A reference to the anonymous is function is returned
var say1 = paint("rose“,"red");
var say2 = paint("sun", "yellow");
alert(say1);
say1();                  // The rose is red;
say2();                   // The sun is yellow;
```



```
function () {
    document.write("<big>" + str + ".</big></br>");
}
```

Arrow (=>) functions

- Instead of the function keyword, it uses an arrow (=>)
- To make it possible to write small function expressions in a less verbose way.

```
const square1 = (x) => { return x * x; };
const square2 = x => x * x;
square2(5); // 25
```

-
  ```
  const sayHello = (e) => {    console.log(`Hello, ${e}`);      };
  sayHello(‘Enrique’); // Hello, Enrique
  ```

arguments object

The arguments of a function are maintained in an array-like object.

```
function myConcat(separator) {
   var result = ‘’;   // initialize list
   var i;             // iterate through arguments
   for (i = 1; i < arguments.length; i++)
             { result += arguments[i] + separator; }
   return result;
}

myConcat(‘, ‘, ‘red’, ‘orange’, ‘blue’);
// returns "red, orange, blue, "

myConcat(‘. ‘, ‘sage’, ‘basil’, ‘oregano’, ‘pepper’, ‘parsley’);
// returns "sage. basil. oregano. pepper. parsley. "
```

Rest parameters

It can be useful for a function to accept any number of arguments. To write such a function, you put three dots before the function's parameter, like this:

```
function max(...numbers) {
  let result = -Infinity;
  for (let number of numbers) {
    if (number > result) result = number;
  }
  return result;
}
console.log(max(4, 1, 9, -2));
// → 9
let numbers = [5, 1, 7];
console.log(max(3, ...numbers,4,8));
// → 8
```

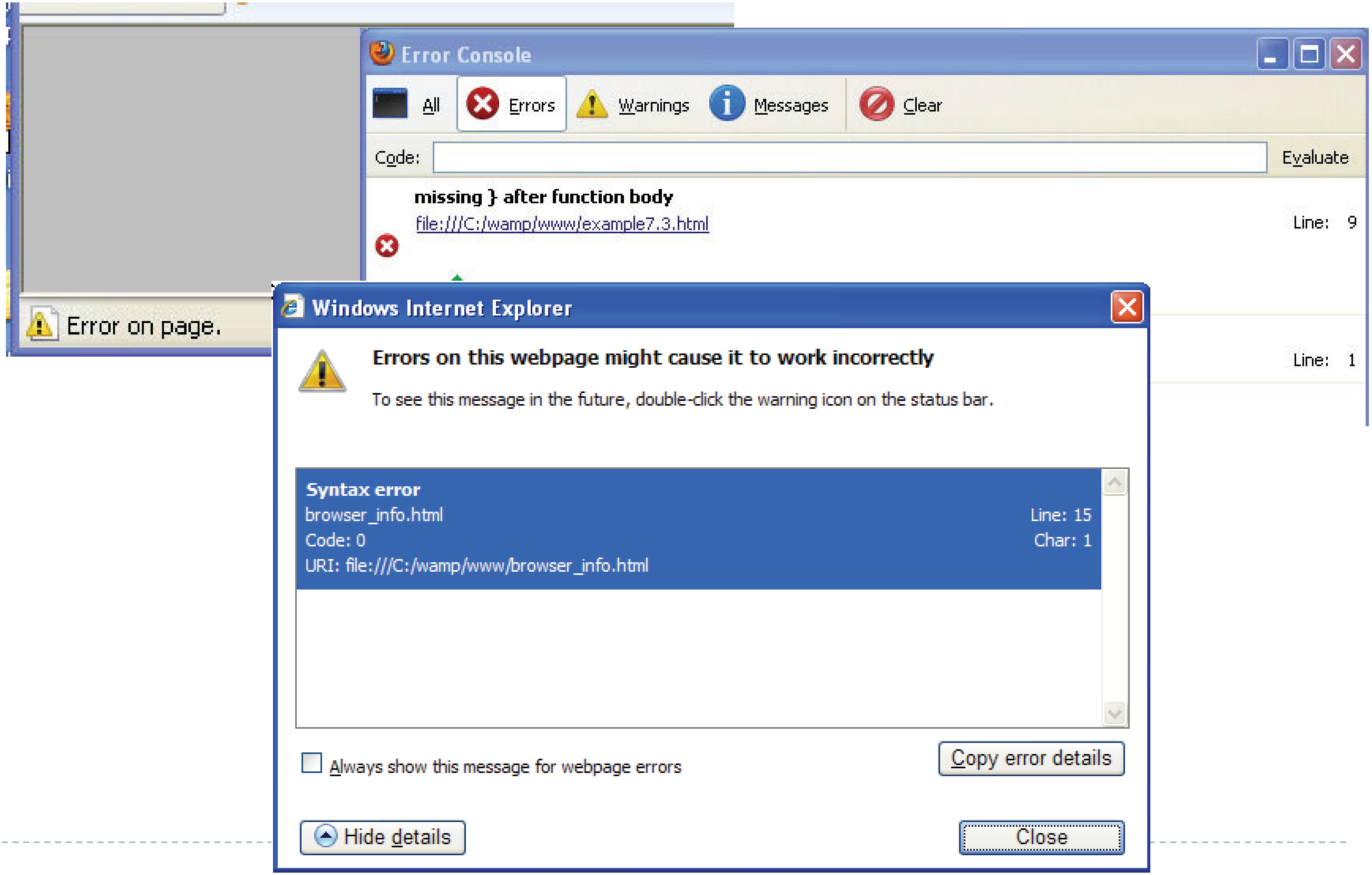
Debugging

Function Syntax

When working with functions there are some simple syntax rules to watch for:

- Did you use parentheses after the function name?
- Did you use opening and closing curly braces to hold the function definition?
- Did you define the function before you called it? Try using the *typeof operator* to see if a function has been defined.
- Did you give the function a unique name?
- When you called the function is your argument list separated by commas? If you don’t have an argument list, did you forget to include the parentheses?
- Do the number of arguments equal to the number of parameters?
- Is the function supposed to return a value? Did you remember to provide a variable or a place in the expression to hold the returned value?
- Did you define and call the function from within a JavaScript program?

Once browser show error message



Error on page.

missing } after function body
file:///C:/wamp/www/example7.3.html
Line: 9

Line: 1

Errors on this webpage might cause it to work incorrectly

To see this message in the future, double-click the warning icon on the status bar.

Syntax error
browser_info.html Line: 15
Code: 0 Char: 1
URI: file:///C:/wamp/www/browser_info.html

□ Always show this message for webpage errors

Exception Handling

- **The *try/catch Statements***

```
<html>
<head><title>Try/Catch</title>
<script type="text/javascript">
try
{
    alert("Current balance is $:" + get_balance());
}
catch(err)
{
    alert("Something went wrong! \n"+
    err.name + ": "+ err.message);
}
</script>
</head>
</html>
```

Error types

Error Name	***When It Is Raised***
EvalError	If the *eval() function is used in* an incorrect manner
RangeError	If a numeric variable or parameter exceeds its allowed range
ReferenceError	If an invalid reference is used; e.g., the variable is undefined
SyntaxError	If a syntax error occurs while parsing code in an *eval()*
TypeError	If the type of a variable or parameter is a valid type
URIError	Raised when encodeURI() or decodeURI() are passed invalid parameters

Objects

What Are Objects?

- JavaScript is all about objects.
  - Windows and buttons, forms and images, links and anchors are all objects.

- Programming languages like Java, C++, and Python that focus on objects are called object-oriented programming (OOP) languages.
-
- JavaScript is called an **object-based** language because it doesn't technically meet the criteria of the more heavy-duty languages, but it certainly behaves as an object-oriented language.

Javascript objects

JavaScript supports several types of objects, as follows:

- User-defined objects defined by the programmer.
- Core or built-in objects, such as Date, String, and Number (
- Browser objects, the BOM (“Browser Objects”).
- The Document objects, the DOM

Creating an Object with a Constructor

General syntax

- var myNewObject = new Object(argument, ...)

To create the *cat object, for example, you could say:*

- var cat = new Object();

JavaScript comes with several built-in constructors

- *Object(), Array(), Date(), and RegExp().*

- var car = new Object();
- var friends = new Array(“Bat", “Bold", “Dorj");
- var holiday = new Date("July 11, 2011");
- var rexp = new RegExp("^[a-zA-Z]");

Properties

What is property?

```
<html>
<head><title>User-defined objects</title>
<script type = "text/javascript">
var toy = new Object();          // Create an instance of the object
toy.name = "Lego";     // Assign properties to the object
toy.color = "red";
toy.shape = "rectangle"; //Properties are not variables. Do not use the var keyword.
</script>
</head>
<body bgcolor="lightblue">
<script type = "text/javascript">
document.write("<b>The toy is a " + toy.name + ".");
document.write("<br />It is a " + toy.color + " " + toy.shape+ ".");
</script>
</body>
</html>
```

In JavaScript you might see the syntax:

- window.document.bgColor = "lightblue";
- bat.math.calc();
- window.close();
- document.write("Hello\n");

User defined object

```
<html><head><title>User-defined objects</title><script type= "text/javascript">
var toy = new Object(); // Create the object
toy.name = "Lego";   // Assign properties to the object
toy.color = "red";
toy.shape = "rectangle";
toy.display=printObject;    // Function name is assigned as a property of the object

function printObject(){
document.write("<b>The toy is a " + toy.name + ".<br>");
document.write("It is a " + toy.color + " " + toy.shape+ ".<br />");
}
</script></head><body><script type = "text/javascript">
toy.display();                       //Object method is called
toy.color="blue";
toy.display();
</script></body></html>
```

What is *this*?

- JavaScript creates an object, and then calls the constructor function. Inside the constructor, the variable *this is initialized to point to this newly created object.*
- *The this keyword is a sort of shorthand reference that keeps track of the current object. For example:*

```
// Create a Book class
function Book(){
this.title = "The White Tiger";      // Create properties
this.author = "Aravind Adiga";
this.Uppage = PageForward;          // Create method
}

function PageForward(){…}

var bookObj = new Book;             // Create new Book object
alert(bookObj.title + " by " + bookObj.author);
bookObj.Uppage();                   //calling method
```

Inline Functions as Methods

- Rather than naming a function outside the class, an inline or anonymous function can be assigned directly to a property within the constructor function.
- Every instance of the class will have a copy of the function code.

```
head…
function Distance(r, t){                    //Constructor function
this.rate = r;
this.time = t;
this.calculate=function() { return r * t; } // anonymous
}
body…
var trip1 = new Distance(50, 1.5);
alert("trip 1 distance: "+ trip1.calculate());
```

Object Literals

- Object literals enable you to create objects that support many features without directly invoking a function.
- When a function acts as constructor you have to keep track of the order of arguments that will be passed, and so on.
- The fields can be nested. The basic syntax for an object literal is:

- A colon (:) separates the property name from its value.
- A comma (,) separates each set of name/value pairs from the next set.
- The comma (,) should be omitted from the last name/value pair. Even with nested key/value pairs, the last key/value pair does not have a comma.
- The entire object is enclosed in curly braces ({ }).

- var object = { property1: value, property2: value };

- The value assigned to a property can be of any data type, including array literals and object literals

Object literal example

```
var soldier = {
name: undefined,
rank: "captain“,
picture: "keeweeboy.jpg“,
fallIn: function() { alert(“Yes sir!");},
…
// Assign value to object property
soldier.name="Tina Savage“;
document.write(“Say ", soldier.name,".<br />");
soldier.fallIn();                    //call object’s method
```

JSON

- { "squirrel": false,
  "events": ["work", "touched tree", "pizza", "running"] }

```
let string = JSON.stringify( { name: "John",  age: 30,  city: "New York" });
console.log(string);
// → { “name”: "John",  “age”: ”30”,  “city”: "New York" }

console.log(JSON.parse(string).city);
// → “New York”
```

The *with* keyword

```
function book(title, author, publisher){
this.title = title; // Properties
this.author = author;
this.publisher = publisher;
this.show = show; // Define a method
}

function show(){
with(this){ // The with keyword with this
var info = "The title is " + title;
info += "\nThe author is " + author;
info += "\nThe publisher is " + publisher;
alert(info);
}}

var childbook = new book(“Book1“, “Book2“,“Book3");
var adultbook = new book(“BookA “,“BookB“,“BookC");
childbook.show(); // Call method for child's book
adultbook.show(); // Call method for adult's book
```

The *for/in* Loop

JavaScript provides the *for/in loop, which can be used to iterate through a list of object* properties or array elements.

```
for(var property_name in object){
statements;
}
```

```
var person={fname:"John", lname:"Doe", age:25};
for (x in person)
   {
   document.write(person[x] + " ");
   }
```

Extending Objects with Prototypes

- JavaScript functions are automatically given an empty *prototype object. If the function* acts as a constructor function for a class of objects, the prototype object can be used to extend the class.
- Each object created for the class is also given two properties, a *constructor* property and a *prototype property.*
- *The* ***constructor*** *property is a reference to the function that* created this object
- The **prototype** property a reference to its prototype object. This property allows the object to share properties and methods.

Adding a new property without using the prototype property

```
function Book(title, author){
this.title =title;
this.author=author;
}
…
var book1 = new Book("Kidnapped","R.L.Stevenson");
var book2 = new Book("Tale of Two Cities", "Charles Dickens")
book1.publisher="Penguin Books";
//A new property, called publisher, is assigned to the book1  object. It is
   available for this instance of the object.
document.write(book1.title + " is published by "+
book1.publisher + "<br />");
document.write(book2.title + " is published by " +
book2.publisher);      //Doesn’t have this property
```

Adding Properties with the Prototype Property

```
function Book(title, author){
this.title =title;
this.author=author;
}
…
var book1 = new Book(“Book1",“Author1");
var book2 = new Book(“Book2",“Author2")

Book.prototype.publisher = "Penguin Books";

alert( book1.title + " is published by“ + book1.publisher );
alert( book2.title + " is published by " + book2.publisher );
```

Creating Subclasses and Inheritance

```
function Pet(){                  // Base Class
      var owner = “Dorj”;       var gender = undefined;
      this.setOwner = function (who) { owner=who;};
      this.getOwner = function ()   { return owner; }
      this.setGender = function (sex) { gender=sex; }
      this.getGender = function ()  { return gender; }  }

function Cat(){}                              //subclass constructor
Cat.prototype = new Pet();                    //all properties and methods of the Pet will now be available to the Cat
Cat.prototype.constructor=Cat;
Cat.prototype.speak= function speak(){ return("Meow"); };

function Dog(){};                //subclass constructor
Dog.prototype= new Pet();
Dog.prototype.constructor=Dog;
Dog.prototype.speak = function speak(){ return("Woof");};

var cat = new Cat;  var dog = new Dog;     cat.setOwner(“Bat");           cat.setGender(“em");            dog.setGender(“er");
alert(cat.getGender() + cat.getOwner() + cat.speak());  alert(dog.getGender() + dog.getOwner() + dog.speak());
```

Properties and Methods of All Objects

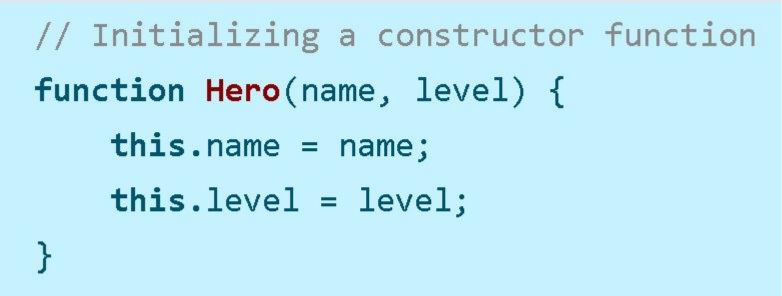
- All user-defined objects and built-in objects are descendants of the object called *Object.*
- *The Object object has its own properties and methods that can be* accessed by any objects derived from it.
- **constructor** A reference to the function that created the object.
- **prototype** A reference to the object prototype for the object. This allows the object to share properties and methods.
- ***toString()*** *Returns a string representing a specified object.*
- ***valueOf()*** *Returns a primitive value for a specified object.*
- ***hasOwnProperty(property)*** *Returns true if the specified property belongs to this object, not* inherited from parent or Object
- ***isPrototypeOf(object)*** *Returns true if this object is one of the parent prototype objects* of the specified child object.

JavaScript Class

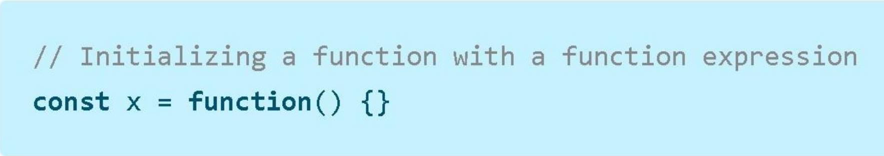
```
class Polygon {
constructor() {
        this.name = "Polygon";  }}

var poly1 = new Polygon();
console.log(poly1.name);
// expected output: "Polygon"
```

Classes Are Functions

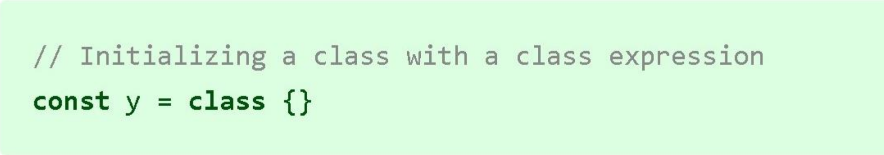
- Classes are declared with the class keyword.



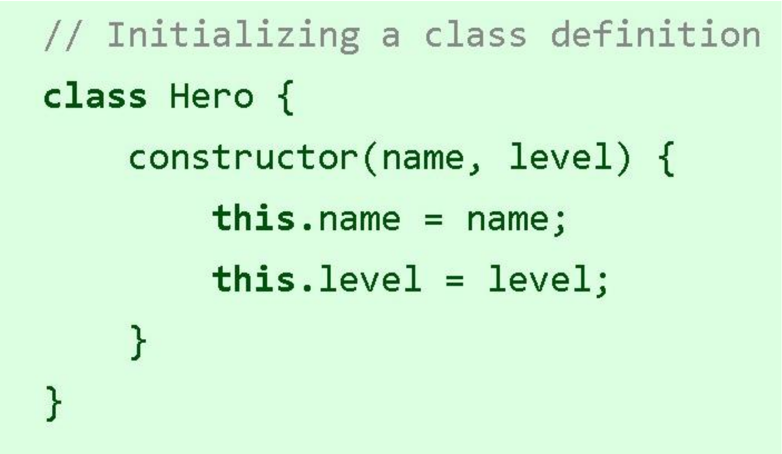
```
// Initializing a constructor function
function Hero(name, level) {
    this.name = name;
    this.level = level;
}
```



```
// Initializing a function with a function expression
const x = function() {}
```



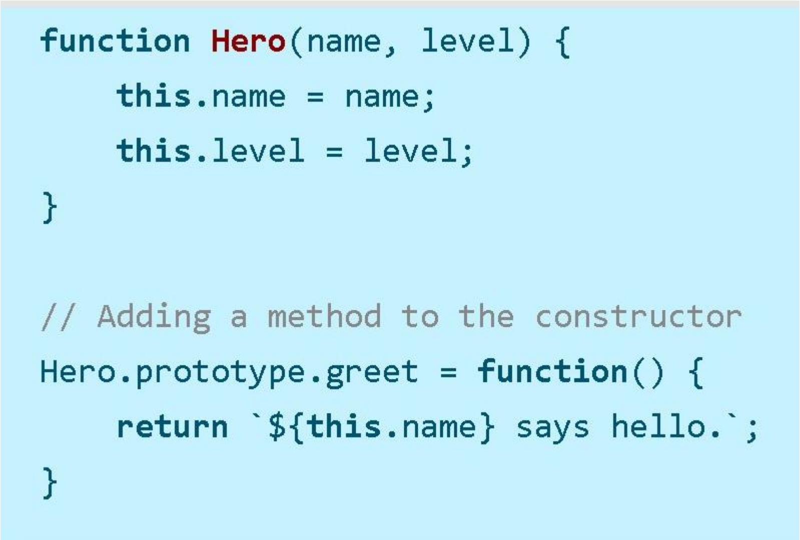
```
// Initializing a class with a class expression
const y = class {}
```



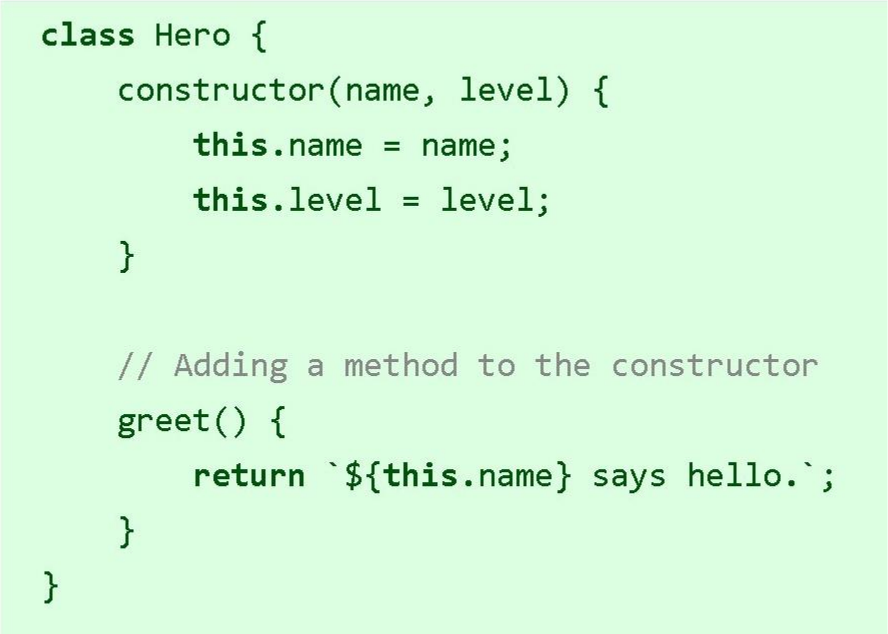
```
// Initializing a class definition
class Hero {
    constructor(name, level) {
        this.name = name;
        this.level = level;
    }
}
```

Defining Methods

- Adding a method



```
function Hero(name, level) {
    this.name = name;
    this.level = level;
}

// Adding a method to the constructor
Hero.prototype.greet = function() {
    return `${this.name} says hello.`;
}
```

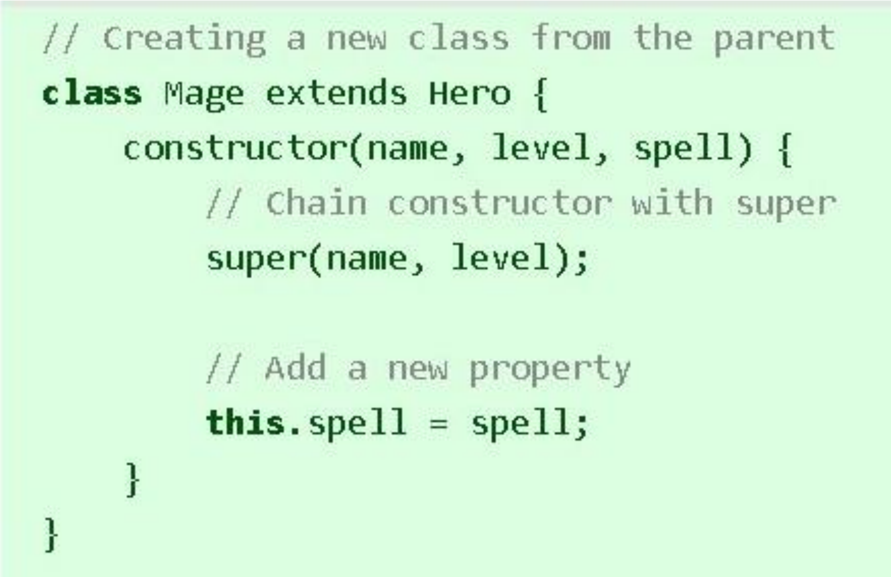


```
class Hero {
    constructor(name, level) {
        this.name = name;
        this.level = level;
    }

    // Adding a method to the constructor
    greet() {
        return `${this.name} says hello.`;
    }
}
```

Extending a class

The call() allows for a function/method belonging to one object to be assigned and called for a different object.

```
function Product(name, price) {
this.name = name;
this.price = price;
}

function Food(name, price) {
  Product.call(this, name, price);
  this.category = 'food’;
}
console.log(new Food('cheese', 5).name);
// expected output: "cheese"
```



```
// Creating a new class from the parent
class Mage extends Hero {
    constructor(name, level, spell) {
        // Chain constructor with super
        super(name, level);

        // Add a new property
        this.spell = spell;
    }
}
```

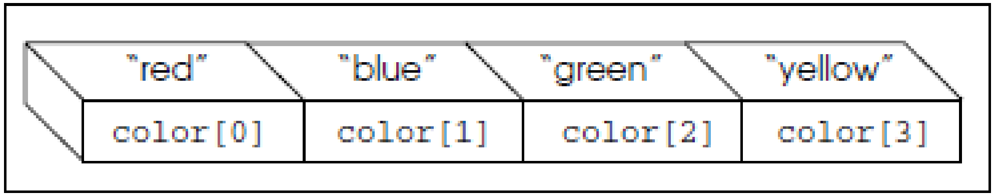
JavaScript Core Objects

- **Array Objects**

- **The *Date Object***
- **The *Math Object***

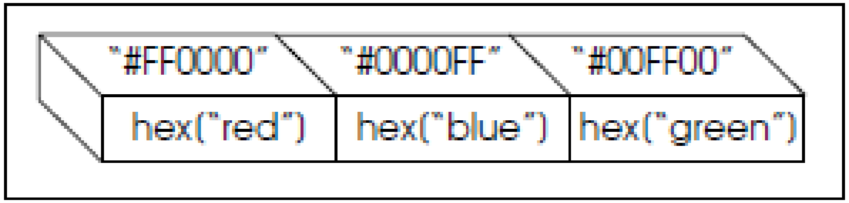
Array objects

- There are two types of index values:
  - a nonnegative integer (**numeric arrays)**



"red"	"blue"	"green"	"yellow"
color[0]	color[1]	color[2]	color[3]

  - string (**associative arrays)**



"#FF0000"	"#0000FF"	"#00FF00"
hex("red")	hex("blue")	hex("green")

Declaring Arrays

- var array_name = new Array();
  - var **months** = new Array();
  - **months**[0]="January";
  - **months**[1]="February";

- var array_name = new Array(100);
- var weekday = new Array("Sun", "Mon", "Tue");
- var myarray=[“Sun",“Mon",“Fri"];

Populating Arrays for/in

```
<html>
<head><title>The Literal Way</title>
<script type="text/javascript">
var pet = [ "Fido", "Slinky", "Tweetie","Wanda" ];
</script>
</head>
<body >
<script type="text/javascript">
for(let i in pet){
console.log("pet[" + i + "] "+ pet[i]);
}
</script>
</body>
</html>
```

Populating Arrays for

```
<script type="text/javascript">
var years = new Array(10);
for(let i=0; i < years.length; i++ )
{
years[i]=i + 2000;
console.log(years[i]);
}
</script>
```

Array properties

- ***constructor*** *References the object's constructor.*
- ***length*** *Returns the number of elements in the array.*
- ***prototype*** *Extends the definition of the array by adding properties and* methods.

Associative Arrays

```
<html><head><title>Associative Arrays</title></head><body>

<script type="text/javascript">
var states = new Array();
states["CA"] = "California";
states["ME"] = "Maine";
states["MT"] = "Montana";

for( let i in states ){
alert("The value is:" + states[i]+ ". ");
}
</script>

</body></html>
```

Bracket vs. Dot Notation ([ ] vs .)

- Any object, not just the *Array* object, can use the square bracket notation to reference it's properties.

- The following two expressions are interchangeable:
  - cat.color = "black";   //suitable for static coding
  - cat["color"] = "black";   //suitable for dynamic coding

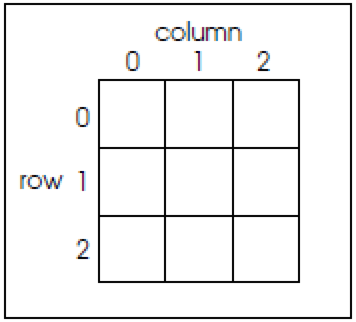
- The bracket notation allows you to use either a string or variable as the index value, whereas the dot notation requires the literal name of the property.

Bracket vs. Dot Notation ([ ] vs .)

```
let myObj = {
  title:“Mr.",
 “first name“ : “Bataa"
};

console.log( myObj[“title"] ); // ?
console.log( myObj.title ); // ?
console.log( myObj.”first name” ); // ?
console.log( myObj[“first name”]); // ?
```

Nested Arrays

- An array can consist of another set of arrays.
- To create a two-dimensional array, each row is a new array.
- To find an element in the array we will use two index values, one for the row and one for the column;
- For example, *array_name[0][0] represents the first element in the first row.The array* consists of three rows and three columns:



Accessing Nested areas

```
var array_name=new Array(
new Array(77,88,99),
new Array(50,60,99),
new Array(99,88,78)
);
```

Set array_name=array_name[0][1]

Array Methods

Method	***What It Does***
concat()	*Concatenates elements from one array to another array.*
join()	*Joins the elements of an array by a separator to form a string.*
pop()	*Removes and returns the last element of an array.*
push()	*Adds elements to the end of an array.*
reverse()	*Reverses the order of the elements in an array.*
shift()	*Removes and returns the first element of an array.*
slice()	*Creates a new array from elements of an existing array.*
sort()	*Sorts an array alphabetically or numerically.*
splice()	*Removes and/or replaces elements of an array.*
toLocaleString()	*Returns a string representation of the array in local format.*
toString()	*Returns a string representation of the array.*
unshift()	*Adds elements to the beginning of an array.*

Map and Set

- The **Map** object holds key-value pairs and remembers the original insertion order of the keys.
- Any value (both objects and primitive values) may be used as either a key or a value.

- The **Set** is a special type collection – “set of values” (without keys), where each value may occur only once.

Map example

```
let things = new Map();
const myFunc = () => '🍕';
things.set('🚗', 'Car');
things.set('🏠', 'House');
things.set('✈️', 'Airplane');
things.set(myFunc, '😁 Key is a function!');
things.size; // 4
things.has('🚗'); // true
things.has(myFunc) // true
things.has(() => '🍕'); // false, not the same reference
things.get(myFunc); // '😁 Key is a function!'
```

```
things.delete('✈️');
things.has('✈️'); // false
things.clear();
things.size; // 0
// setting key-value pairs is chainable
things.set('🔧', 'Wrench')
    .set('🎸', 'Guitar')
    .set('🕹️', 'Joystick');
const myMap = new Map();
// Even another map can be a key
things.set(myMap, 'Oh gosh!');
things.size; // 4
things.get(myMap); // 'Oh gosh!'
```

Map iteration

```
let activities = new Map();
activities.set(1, '🏂');
activities.set(2, '🏎');
activities.set(3, '🚣’);

for (let [nb, activity] of activities) {
  console.log(`Activity ${nb} is ${activity}`);
}
activities.forEach((value, key) => { console.log(`Activity ${key} is ${value}`); });

// Activity 1 is 🏂
// Activity 2 is 🏎
// Activity 3 is 🚣
```

Set example

```
let set = new Set();
let john = { name: "John" };
let pete = { name: "Pete" };
// visits, some users come multiple times
set.add(john);
set.add(pete);
set.add(john);
// set keeps only unique values
alert( set.size ); // 2
for (let user of set) {
  alert(user.name); // John (then Pete)
}
```

Set iteration

```
let myAnimals = new Set(['🐷', '🐢', '🐷', '🐷']);

myAnimals.add(['🐨', '🐑']);
myAnimals.add({ name: 'Rud', type: '🐢' });
console.log(myAnimals.size); // 4

myAnimals.forEach(animal => {
  console.log(animal);
});


// 🐷
// 🐢
// ["🐨", "🐑"]
// Object { name: "Rud", type: "🐢" }
```

Date Object

Format

- new Date("Month dd, yyyy hh:mm:ss")
- new Date("Month dd, yyyy")
- new Date(yy,mm,dd,hh,mm,ss)
- new Date(yy,mm,dd)
- new Date(milliseconds)

Examples of instantiating a date:

- mydate = new Date()
- mydate = new Date("March 15, 2010 09:25:00")
- mydate = new Date(“March 15, 2010”)
- mydate = new Date(10,2,15)
- mydate = new Date(10,2,15,9,25,0)
- mydate = new Date(500);

- **var now = new Date(); *// Now is an instance of a Date object***
- document.write("<b>Local time:</b> " + now + "<br />");
- **var hours=now.getHours();**
- **var minutes=now.getMinutes();**
- **var seconds=now.getSeconds();**
- **var year=now.getFullYear();**

Math Object

- *Math.abs(Number) Returns the absolute (unsigned) value of Number*
- *Math.acos(Number) Arc cosine of Number, returns result in radians*
- *Math.asin(Number) Arc sine of Number, returns results in radians*
- *Math.atan(Number) Arctangent of Number, returns results in radians*
- *Math.atan2(y,x) Arctangent of y/x; returns arctangent of the quotient of its* arguments
- ***Math.ceil(Number)*** *Rounds Number up to the next closest integer*
- *Math.cos(Number) Returns the cosine of Number in radians*
- *Math.exp(x)* Euler's constant to some power*
- ***Math.floor(Number)*** *Rounds Number down to the next closest integer*
- *Math.log(Number) Returns the natural logarithm of Number (base E)*
- *Math.max(Number1, Number2) Returns larger value of Number1 and Number2*
- *Math.min(Number1, Number2) Returns smaller value of Number1 and Number2*
- *Math.pow(x, y) Returns the value of x to the power of y(xy), where x is the* base and *y is the exponent*
- *Math.random() Generates pseudorandom number between 0.0 and 1.0*
- ***Math.round(Number)*** *Rounds Number to the closest integer*
- *Math.sin(Number) Arc sine of Number in radians*
- *Math.sqrt(Number) Square root of Number*

Rounding Up and Rounding Down

Number	***ceil()***	***floor()***	***round()***
2.55	3	2	3
2.30	3	2	2
–2.5	–2	–3	–2
–2.3	–2	–3	–2