**JavaScript**

|  |  |
| --- | --- |
| **== and ===**  == : does automatic type conversion  === : is called strict equality operator. The value must be equal in type as well  0 == false is true where as 0 === false is false  null == undefined is true where as null === undefined is false  **DataType of Variables**  All variables are OBJECT DataType  **Escape Character**  Backslash \  used when working with special characters like single quote, double quote, apostrophes and ampersands  **Generic Objects**  var i = new object();  **Null And Undefined**  null : explicitly been given null value. It is an object so if bar == null then bar == object is true. We'll need to check for null here as well.  undefined : variable has been declared but no value assigned  **Global Variables**  Declared without the keyword var or assigning a value without declaring it  The context is windows  var x = 5 local variable  y = 6 is global variable  Any function including the windows object can overwrite these  **Strict Mode**  "Use Strict"  The code will raise an error if variable is undefined  Error is UncaughtReferenceError  var a = b = 5 will raise an error  var a = windows.b = 5 this is ok  **Create Native Methods**  Native javascript objects store their methods in prototype (extensions)  console.log('hello'.repeatify(3));  Should print hellohellohello.  String.prototype.repeatify = String.prototype.repeatify || function(times) {  var str = '';  for (var i = 0; i < times; i++) {  str += this;  }  return str;  };  **Closure / Wrap Code inside Function Block**  This creates a private name space. Namespacing is grouping the desired functions variables etc under a unique name.  Helps avoid potential name clashes between different javascript modules and libraries  <script>  var add = (function(x){  return (function(y){  return x + y;  })(2);  })(1);  //////////////////////////  var counter = (function(){  var counter = 0;  //this is also called self-invoking function, only runs once  return function(){  return counter += 1;  }  })();  function addCounter(){  altert(counter);  }  addCounter() : 1  addCounter() : 2  addCounter() : 3  </script>  **Window.OnLoad and onDocumentReady**  The onload function is not run till all the information on the page has loaded  Where as onDocumentReady loads the code just after the DOM has loaded  **Anonymous Function**  without a named identifier  in-accessible after its declaration  <script>  var someThing = function(){  alert("I am anonymous");  };  someThing();  </script>  **Escape / Unescape**  Escape codes the string  Unescape decodes the string  **Securing Websites**   * Keep servers up-to-date with patches * Keep third party SW up-to-date like CMS * XSS – cross site scripting. Make sure to validate or take out any HTML or encode HTML * Do not pin point error messages * Validation * Strong passwords * SSL | **Event Bubbling**  DOM elements can be nested inside each other, the handler of the parent works even when you click on the child  <div class="d1" onclick="highlight(this)">1  <div class="d2" onclick="highlight(this)">2  <div class="d3" onclick="highlight(this)">3</div>  </div>  </div>  <script>  function highlight(elem) {  elem.style.backgroundColor='yellow'  alert(elem.className)  elem.style.backgroundColor = ''  }  </script>  **Popup Windows**  Alert  Confirm  Prompt  **THIS**  THIS is the object that "owns" the current code  It is the current object in the code  <script>  var object = {  firstName: 'john',  lastName: 'Doe',  fullName: function(){  return this.firstName + ' ' + this.lastName;  }  }    alert(object.fullName()); //will print John Doe  </script>  **NaN**  Not a number, its type is number  User number.isNan() to reliably check  **parseFloat / parseInt**  parseFloat takes single string value  parseInt takes string and redix  **JavaScript Types**  number, string, boolean, function, null, object, undefined  **Negative Infinity**  a number derived by dividing negative number by zero  **Timers**  setTimeOut, setInterval, and clearInterval  To run a code at specified time or repeat code in a given interval  **Looping Structures**  For, While, Do-While  **Delete Operator**  It is basically a remove  deletes all variables and objects in a program.  it cannot delete variables declared with a key word var  <script>  var set = {};  set.x = true;  delete set.x;  </script>  **Break and Continue**  Break statement exits from the current loop  Continue statement continues with the next statement of the loop  **Errors in JavaScript**  Load Time Errors: due to improper syntax like missing opening or closing brackets. Happens at load time.  Run Time Errors: happens due to mis-use of the command -- spelling mistakes for variable names  Logical Errors: happens due to bad logic, may be due to invalid data  **How are Object Properties Assigned**  In the same way as a value is assigned to a variable  Document.form.action = "submit"  **Screen Objects**  to read information from clients screen  AvalHeight: Gives the height of client’s screen  AvailWidth: Gives the width of client’s screen.  ColorDepth: Gives the bit depth of images on the client’s screen  Height: Gives the total height of the client’s screen, including the taskbar  Width: Gives the total width of the client’s screen, including the taskbar  **Encode URL / Decode URL**  Encode URL convers the url into hex coding  Decode URL converts the url back to normal  **SQL Injection**   * Weak input validation * Dynamic construction of SQL statements without the use of type safe parameters * Use of over privileged DB logins |

**Inheritance in Javascript**

We have class inheritance (with call) and prototype inheritance with prototype chaining.

**Function 1**: **simple with properties (Single Point) - called an object initializer or an object literal has 4 properties**

|  |  |
| --- | --- |
| var point = {  x: 5,  y: 2,  dist: function () {  return Math.sqrt((this.x\*this.x)+(this.y\*this.y));  },  toString: function () {  return "("+this.x+", "+this.y+")";  }  }; | var r = 'x=' + point.x + '<br>y=' + point.y + '<br>dist=' + point.dist() + '<br>toStrig=' + point.toString()  x=5  y=2  dist=5.385164807134504  toStrig=(5, 2) |

**If we need several points, we need a factory of objects, in JS are called constructors**

**Function 2: methods in the instance – with constructor**

|  |  |
| --- | --- |
| function Point(x, y) {  this.x = x;  this.y = y;  this.dist = function () {  return Math.round(Math.sqrt((this.x\*this.x)+(this.y\*this.y)) \* 100) / 100 ;  };  this.toString = function () {  return "("+this.x+", "+this.y+")";  };  } | var p = new Point(2,4);  var r = 'x=' + p.x + '<br>y=' + p.y + '<br>dist=' + p.dist() + '<br>toStrig=' + p.toString()  x=2  y=4  dist=4.47  toStrig=(2, 4) |

**Function 3:** **methods should not be in each instance, these should be shared**

|  |  |
| --- | --- |
| function Point(x, y) {  this.x = x;  this.y = y;  }  Point.prototype = {  dist: function () {  return Math.round(Math.sqrt((this.x\*this.x)+(this.y\*this.y)) \* 100) / 100;  },  toString: function () {  return "("+this.x+", "+this.y+")";  }  } | var p = new Point(2,4);  var r = 'x=' + p.x + '<br>y=' + p.y + '<br>dist=' + p.dist() + '<br>toStrig=' + p.toString()  x=2  y=4  dist=4.47  toStrig=(2, 4) |

**Function 4**: **same as 3 but** **dist and toString getting their own prototype**

|  |  |
| --- | --- |
| function Point(x, y) {  this.x = x;  this.y = y;  }  Point.prototype.dist = function () {  return Math.round(Math.sqrt((this.x\*this.x)+(this.y\*this.y)) \* 100) / 100;  };  Point.prototype.toString = function () {  return "("+this.x+", "+this.y+")";  }; | var p = new Point(2,4);  var r = 'x=' + p.x + '<br>y=' + p.y + '<br>dist=' + p.dist() + '<br>toStrig=' + p.toString()  x=2  y=4  dist=4.47  toStrig=(2, 4) |

**Inheritance**

JS is a class-free, object-oriented language, and as such, it uses prototypal inheritance instead of classical inheritance

Above means an object inherits from another object

We can use Call() function but again its prototype inheritance

**Function 5:**

|  |  |
| --- | --- |
| **simple classes**  //base object  var employee = function(name){  this.name = name;  }  employee.prototype.getName = function(){  return this.name;  }  //manager will inherit employee  var manager = function(annualSalary){  this.annualSalary = annualSalary;  }  manager.prototype.getSalary = function(){  return this.annualSalary;  }  var emp = new employee('xyz');  var man = new manager(140);  jQuery( "#result" ).html( 'name=' + emp.getName() + '<br>salary=' + man.getSalary() ) ;  name=xyz  salary=140 | **Simple inheritance,** we will associate employee as the parent of manager. hasOwnProperty given us false if the property is on prototype.  //base object  var employee = function(name){  this.name = name;  }  employee.prototype.getName = function(){  return this.name;  }  employee.prototype.getNameLength = function(){  return this.name.length;  }  //manager will inherit employee  var manager = function(annualSalary){  this.annualSalary = annualSalary;  }  //create an object of type employee  var emp = new employee('xyz');  //we will mark the above object as parent of manager **or new emp**  manager.prototype = emp;  //create an instance of manager  var man = new manager(140);  //man.name = "abc"; //xyz will be over written  jQuery( "#result" ).html( 'name=' + man.getName() + '<br>length:' + man.getNameLength() + '<br>salary=' + man.annualSalary ) ;  name=xyz  length:3  salary=140 |

**Function 6: another example with using call() function.**

* Call let us call a function defined somewhere else, but in the current context.
* Techer is inheriting from person
* We need to get teacher() to inherit the methods from person.

|  |  |
| --- | --- |
| **Simple classes**  var person = function(first, last, age, gender){  this.name = {  first,  last  };  this.age = age;  this.gender = gender;  }  person.prototype.greeting = function() {  return 'Hi! I\'m ' + this.name.first + '.';  };  var p = new person('abc', 'xyz', 21, 'M');  jQuery( "#result" ).html( 'first:' + p.name.first + '<br>last:' + p.name.last + '<br>age:' + p.age + '<br>gender:' + p.gender + '<br>greeting:' + p.greeting());  first:abc  last:xyz  age:21  gender:M  greeting:Hi! I'm abc. | **Simple inheritance**  //person class  var person = function(first, last, age, gender){  this.name = {  first,  last  };  this.age = age;  this.gender = gender;  }  //teacher  var teacher = function(first, last, age, gender, subject) {  //call let us call the function defined some where else  person.call(this, first, last, age, gender);  this.subject = subject;  }  //let teacher inherit the person  teacher.prototype = Object.create(person.prototype);  teacher.prototype.constructor = teacher;  //person greeting  person.prototype.greeting = function() {  return 'Hi! I\'m ' + this.name.first + '.';  };  //teacher greeting  teacher.prototype.greeting2 = function(){  var prefix;  if(this.gender === 'M' || this.gender === 'm')  prefix = 'Mr.';  else if(this.gender === 'F' || this.gender === 'f')  prefix = "Mrs.";  else  prefix = "Mx.";  return 'Hi! I\'m ' + prefix + ' ' + this.name.first + '.';  };  var t1 = new teacher('def', 'iop', 25, 'm', 'physics')  jQuery( "#result" ).html( 'first:' + t1.name.first + '<br>last:' + t1.name.last + '<br>age:' + t1.age + '<br>gender:' + t1.gender + '<br>greeting:' + t1.greeting() + '<br>subject:' + t1.subject+'<br>greeting2:'+t1.greeting2());  first:def  last:iop  age:25  gender:m  greeting:Hi! I'm def.  subject:physics  greeting2:Hi! I'm Mr. def. |