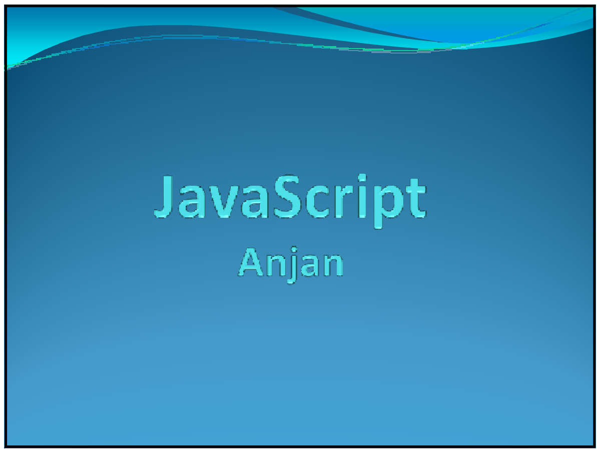
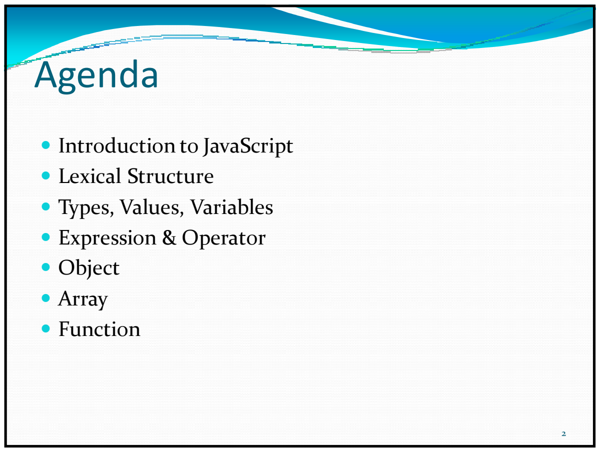


JavaScript

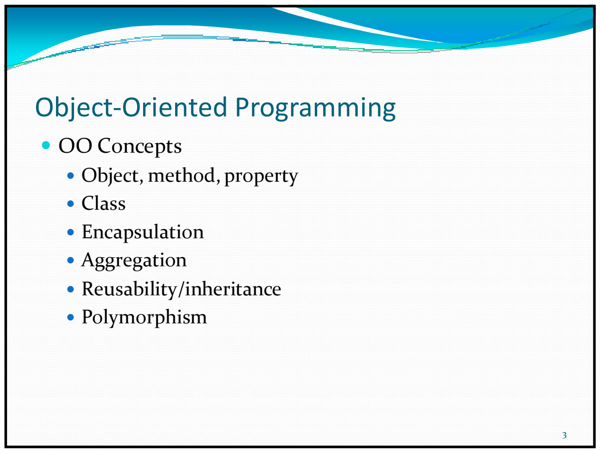
Anjan



Agenda

- Introduction to JavaScript
- Lexical Structure
- Types, Values, Variables
- Expression & Operator
- Object
- Array
- Function

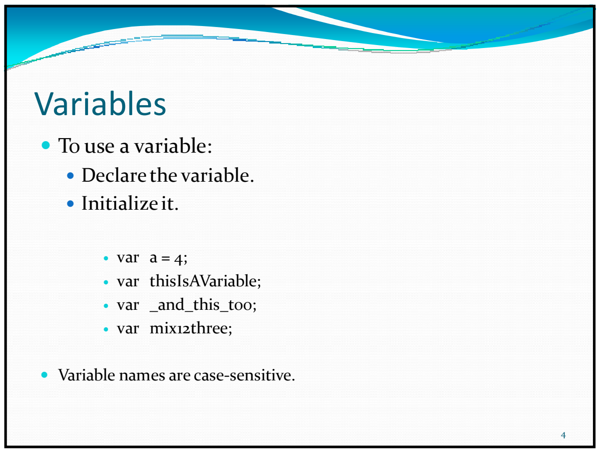
2



Object-Oriented Programming

- OO Concepts
  - Object, method, property
  - Class
  - Encapsulation
  - Aggregation
  - Reusability/inheritance
  - Polymorphism

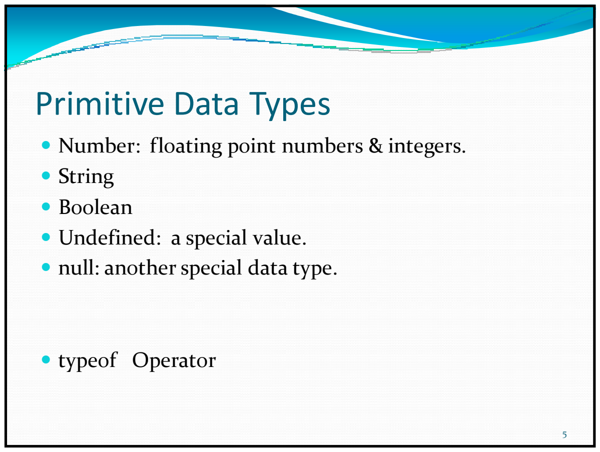
3



Variables

- To use a variable:
  - Declare the variable.
  - Initialize it.
    - var a = 4;
    - var thisIsAVariable;
    - var _and_this_too;
    - var mix12three;
- Variable names are case-sensitive.

4

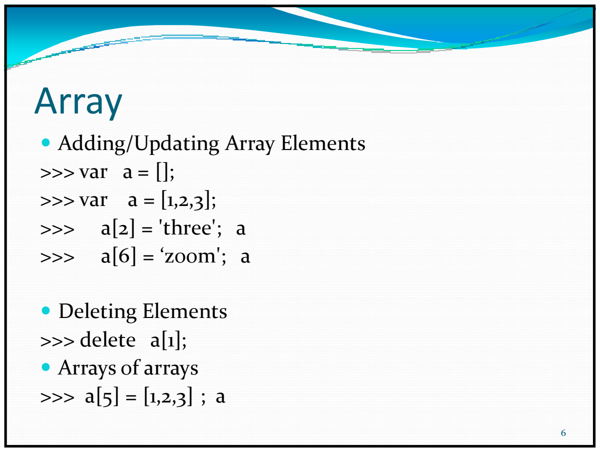


Primitive Data Types

- Number: floating point numbers & integers.
- String
- Boolean
- Undefined: a special value.
- null: another special data type.

- typeof Operator

5



Array

- Adding/Updating Array Elements

```
>>> var   a = [];
>>> var    a = [1,2,3];
>>>     a[2] = 'three';   a
>>>     a[6] = ‘zoom';   a
```

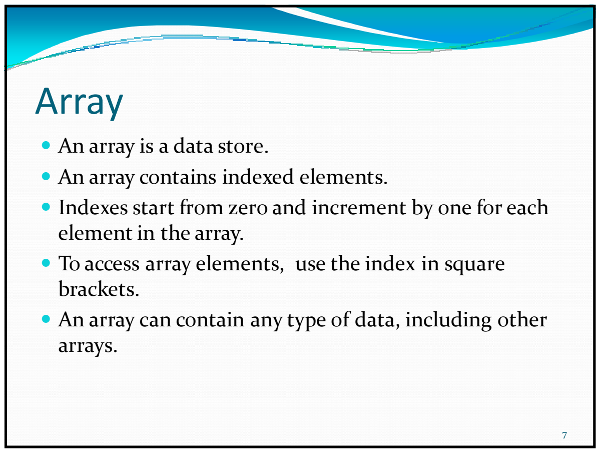
- Deleting Elements

```
>>> delete   a[1];
```

- Arrays of arrays

```
>>>  a[5] = [1,2,3]  ;  a
```

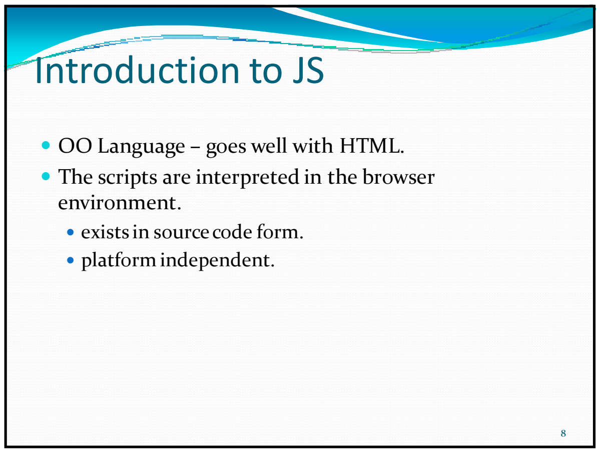
6



Array

- An array is a data store.
- An array contains indexed elements.
- Indexes start from zero and increment by one for each element in the array.
- To access array elements, use the index in square brackets.
- An array can contain any type of data, including other arrays.

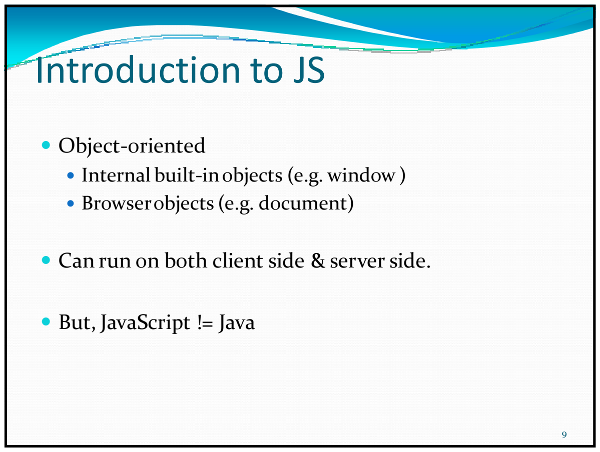
7



Introduction to JS

- OO Language – goes well with HTML.
- The scripts are interpreted in the browser environment.
  - exists in source code form.
  - platform independent.

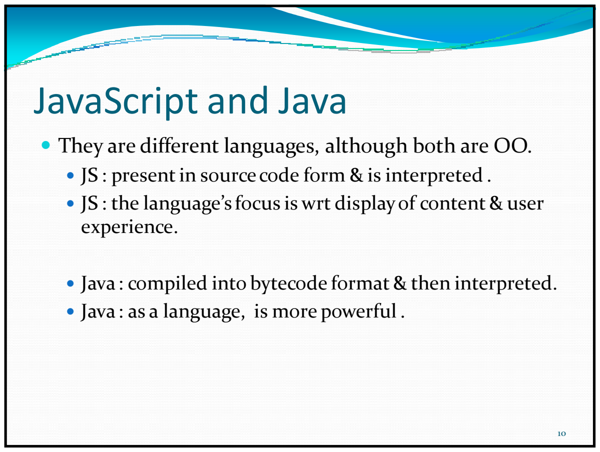
8



Introduction to JS

- Object-oriented
  - Internal built-in objects (e.g. window)
  - Browser objects (e.g. document)
- Can run on both client side & server side.
- But, JavaScript != Java

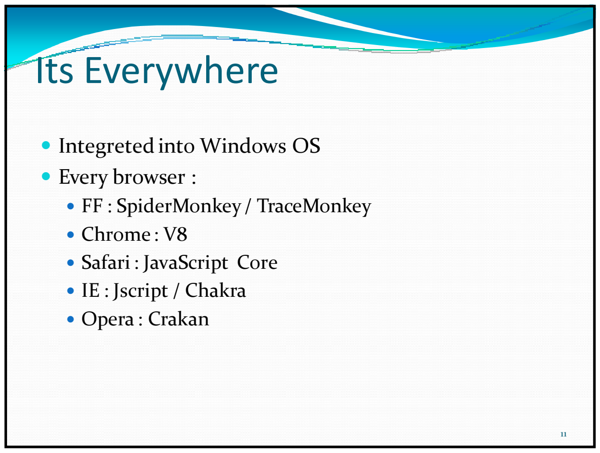
9



JavaScript and Java

- They are different languages, although both are OO.
  - JS : present in source code form & is interpreted .
  - JS : the language's focus is wrt display of content & user experience.
  - Java : compiled into bytecode format & then interpreted.
  - Java : as a language, is more powerful .

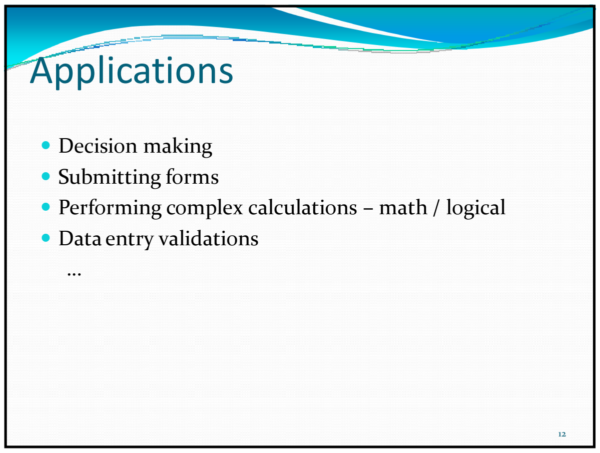
10



Its Everywhere

- Integreted into Windows OS
- Every browser :
  - FF : SpiderMonkey / TraceMonkey
  - Chrome : V8
  - Safari : JavaScript Core
  - IE : Jscript / Chakra
  - Opera : Crakan

11

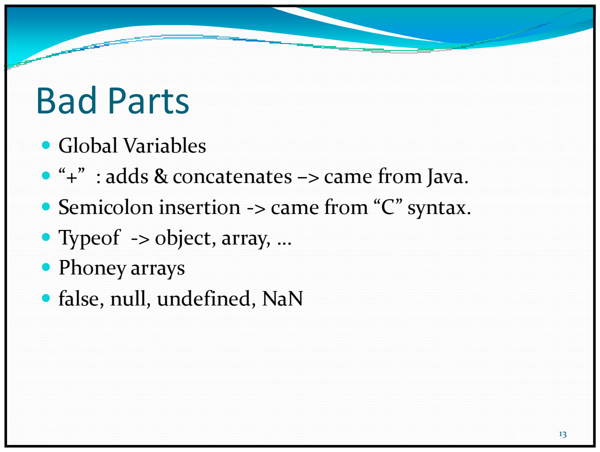


Applications

- Decision making
- Submitting forms
- Performing complex calculations – math / logical
- Data entry validations

  …

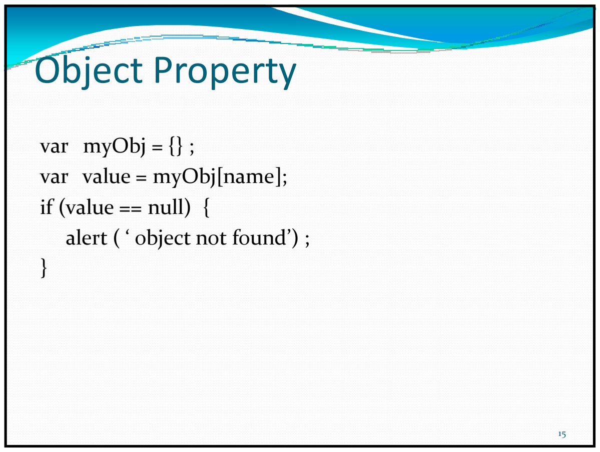
12



Bad Parts

- Global Variables
- "+" : adds & concatenates –> came from Java.
- Semicolon insertion -> came from "C" syntax.
- Typeof -> object, array, …
- Phoney arrays
- false, null, undefined, NaN

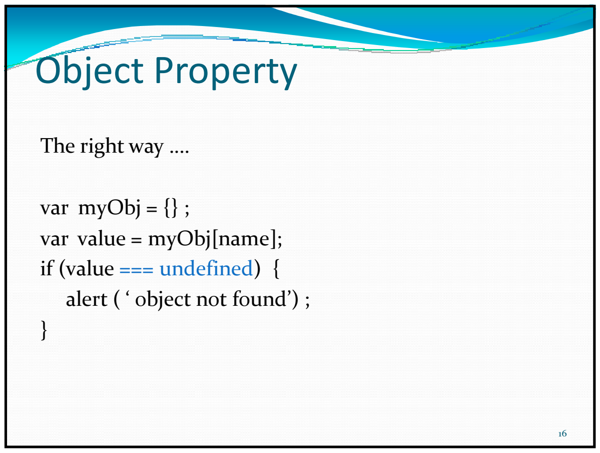
13



Object Property

```
var   myObj = {} ;
var   value = myObj[name];
if (value == null)  {
     alert ( ' object not found') ;
}
```

15

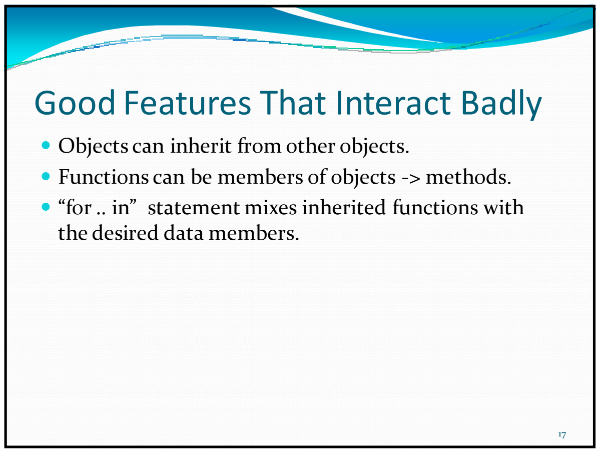


Object Property

The right way ….

```
var  myObj = {} ;
var  value = myObj[name];
if (value === undefined)  {
     alert ( ' object not found') ;
}
```

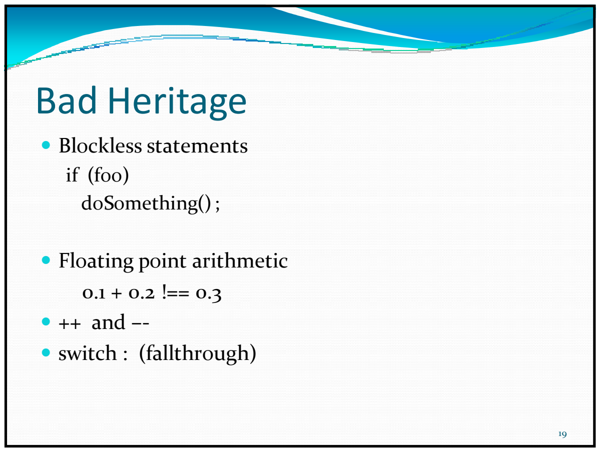
16



Good Features That Interact Badly

- Objects can inherit from other objects.
- Functions can be members of objects -> methods.
- "for .. in" statement mixes inherited functions with the desired data members.

17



Bad Heritage

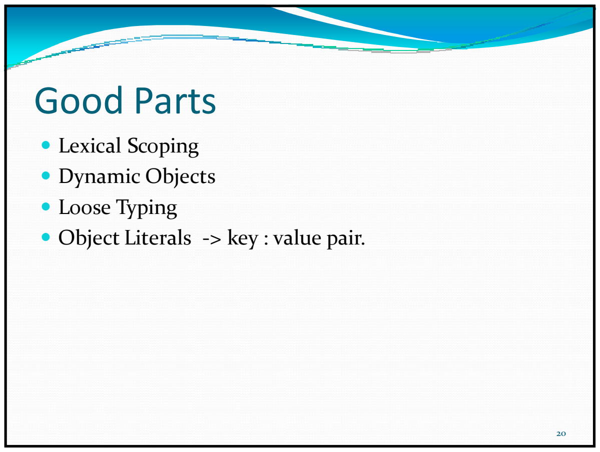
- Blockless statements

```
if (foo)
  doSomething() ;
```

- Floating point arithmetic

  0.1 + 0.2 !== 0.3
- ++ and --
- switch : (fallthrough)

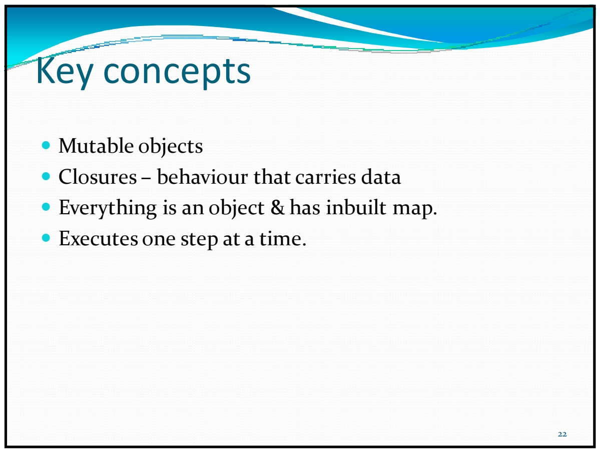
19



Good Parts

- Lexical Scoping
- Dynamic Objects
- Loose Typing
- Object Literals -> key : value pair.

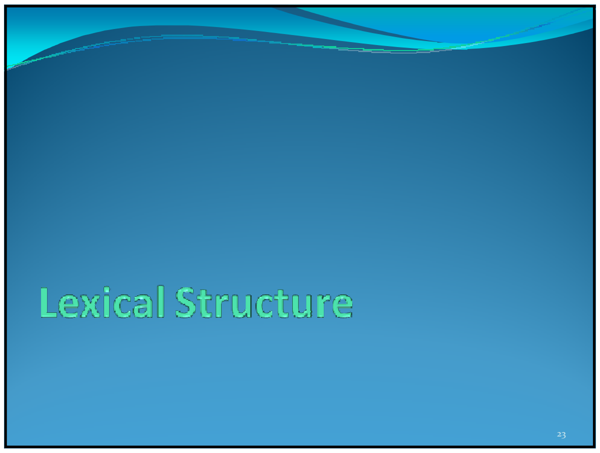
20



Key concepts

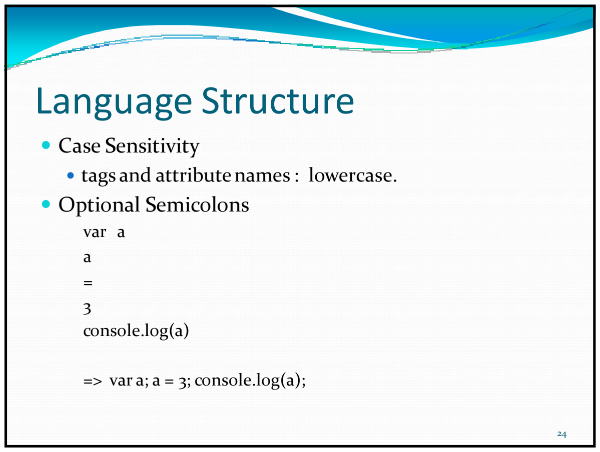
- Mutable objects
- Closures – behaviour that carries data
- Everything is an object & has inbuilt map.
- Executes one step at a time.

22



Lexical Structure

23

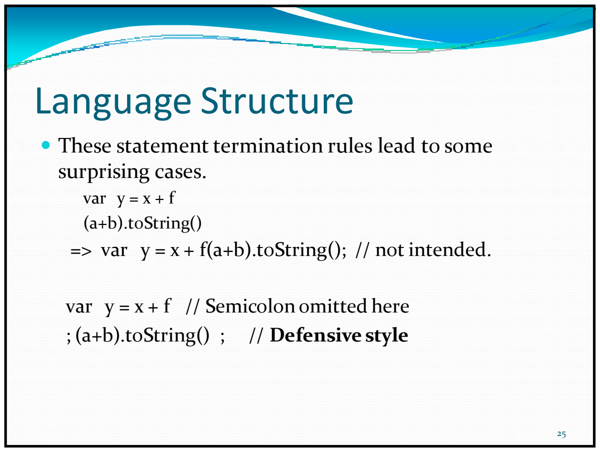


Language Structure

- Case Sensitivity
  - tags and attribute names : lowercase.
- Optional Semicolons

```
var  a
a
=
3
console.log(a)

=>  var a; a = 3; console.log(a);
```

24

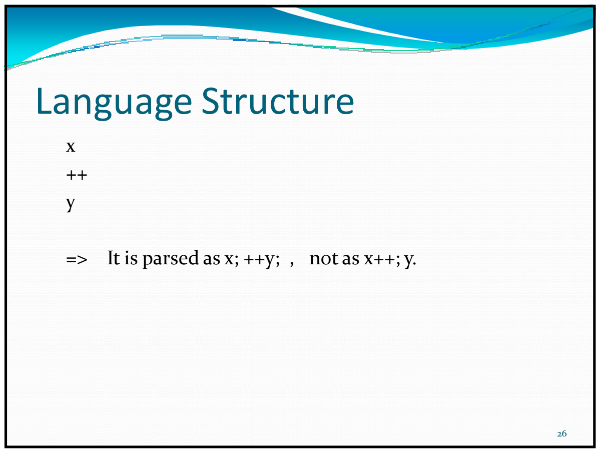


Language Structure

- These statement termination rules lead to some surprising cases.

```
    var  y = x + f
    (a+b).toString()
  =>  var  y = x + f(a+b).toString();  // not intended.

  var  y = x + f   // Semicolon omitted here
  ; (a+b).toString()  ;    // Defensive style
```

25



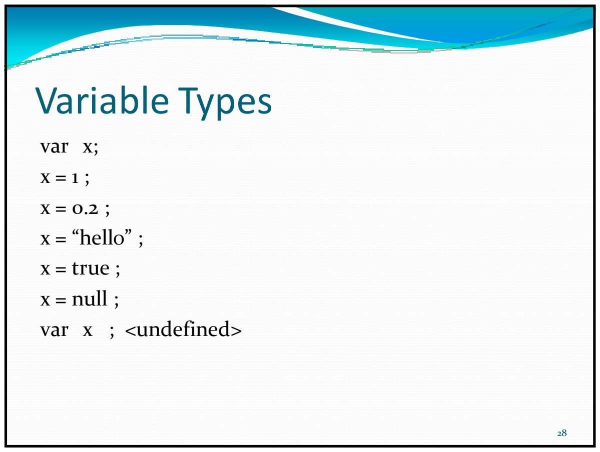
Language Structure

```
x
++
y

=>    It is parsed as x; ++y;  ,   not as x++; y.
```

26

Types, Values, Variables

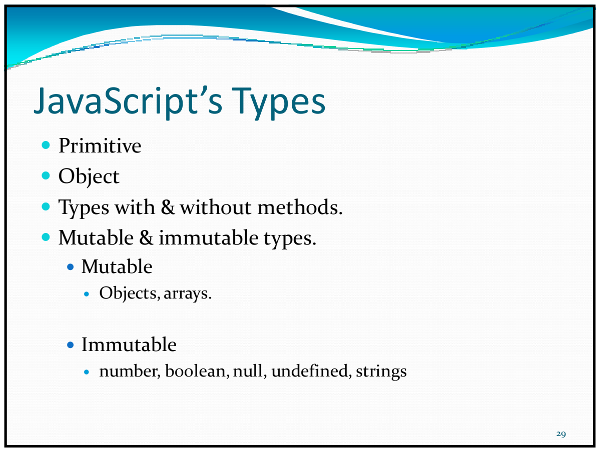
27



Variable Types

```
var   x;
x = 1 ;
x = 0.2 ;
x = “hello” ;
x = true ;
x = null ;
var   x   ;  <undefined>
```

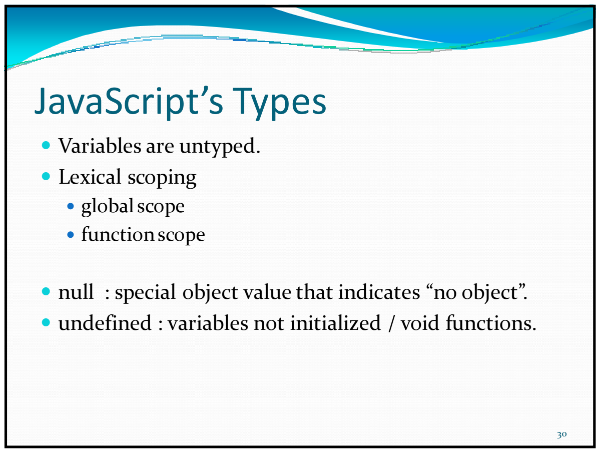
28



JavaScript’s Types

- Primitive
- Object
- Types with & without methods.
- Mutable & immutable types.
  - Mutable
    - Objects, arrays.
  - Immutable
    - number, boolean, null, undefined, strings

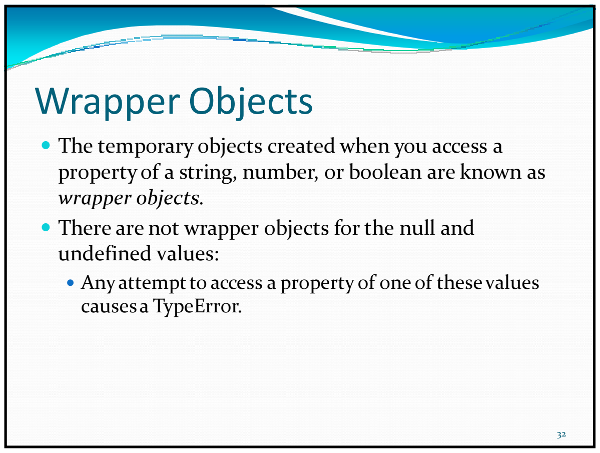
29



JavaScript’s Types

- Variables are untyped.
- Lexical scoping
  - global scope
  - function scope
- null : special object value that indicates “no object”.
- undefined : variables not initialized / void functions.

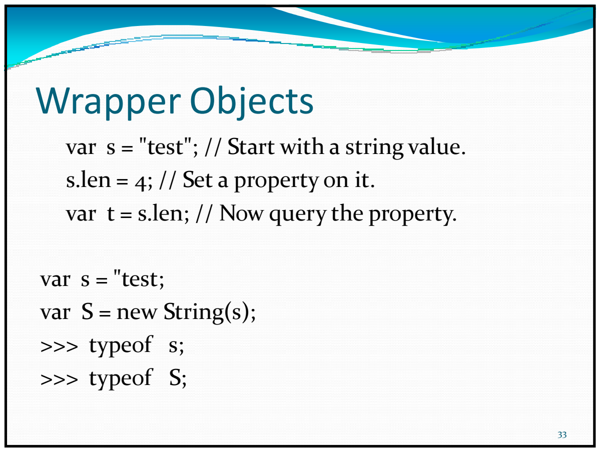
30



Wrapper Objects

- The temporary objects created when you access a property of a string, number, or boolean are known as *wrapper objects.*
- There are not wrapper objects for the null and undefined values:
  - Any attempt to access a property of one of these values causes a TypeError.

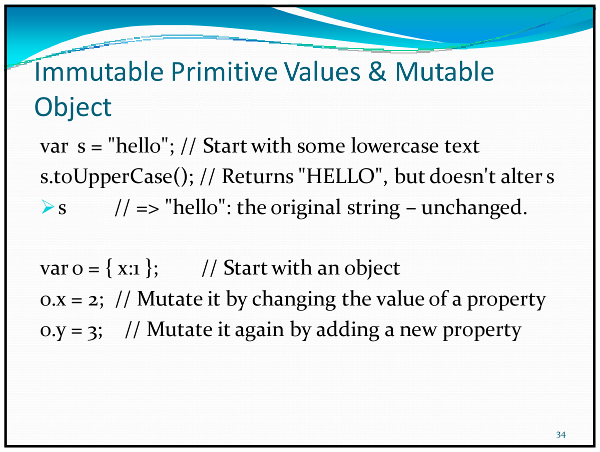
32



Wrapper Objects

```
var  s = "test"; // Start with a string value.
s.len = 4; // Set a property on it.
var  t = s.len; // Now query the property.

var  s = "test;
var  S = new String(s);
>>>  typeof   s;
>>>  typeof   S;
```

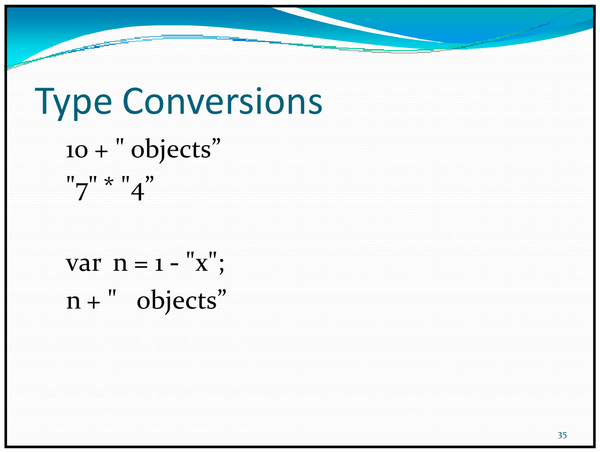
33



Immutable Primitive Values & Mutable Object

```
var  s = "hello"; // Start with some lowercase text
s.toUpperCase(); // Returns "HELLO", but doesn't alter s
➢s        // => "hello": the original string – unchanged.

var o = { x:1 };        // Start with an object
o.x = 2;  // Mutate it by changing the value of a property
o.y = 3;    // Mutate it again by adding a new property
```

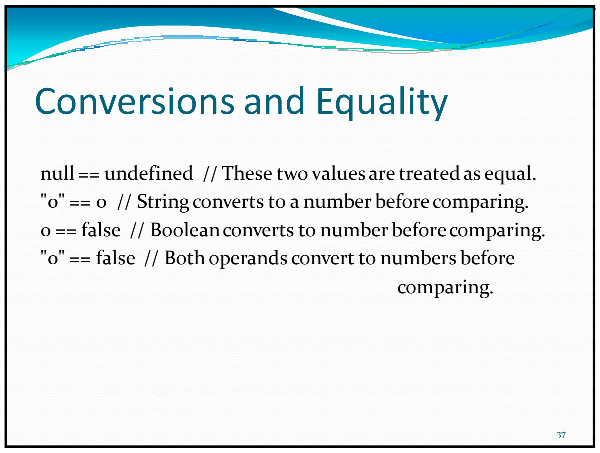
34



Type Conversions

```
10 + " objects”
"7" * "4”

var  n = 1 - "x";
n + "   objects”
```

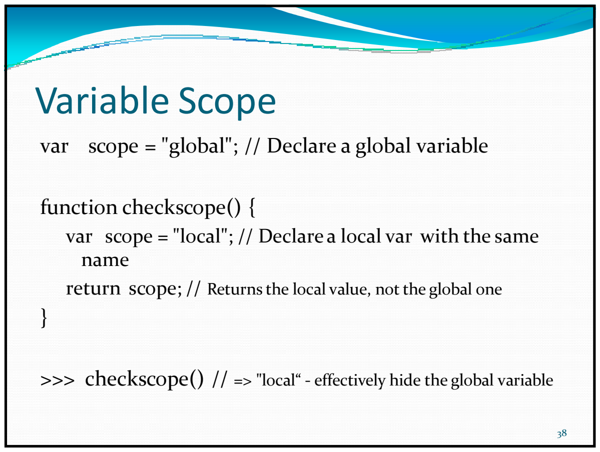
35



Conversions and Equality

```
null == undefined  // These two values are treated as equal.
"0" == 0  // String converts to a number before comparing.
0 == false  // Boolean converts to number before comparing.
"0" == false  // Both operands convert to numbers before
                                                   comparing.
```

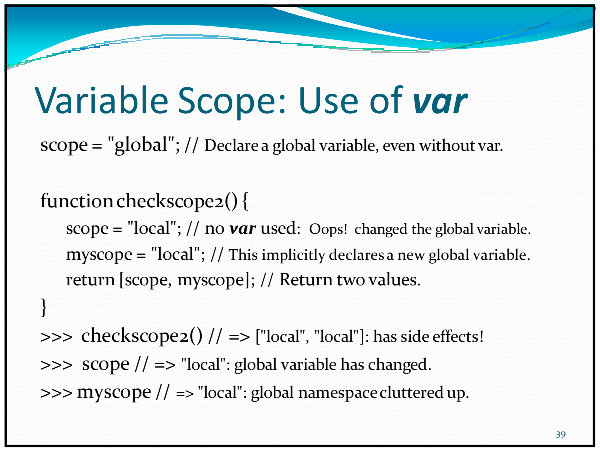
37



Variable Scope

```
var   scope = "global"; // Declare a global variable

function checkscope() {
    var  scope = "local"; // Declare a local var  with the same
       name
    return  scope; // Returns the local value, not the global one
}

>>>  checkscope()  // => "local“ - effectively hide the global variable
```

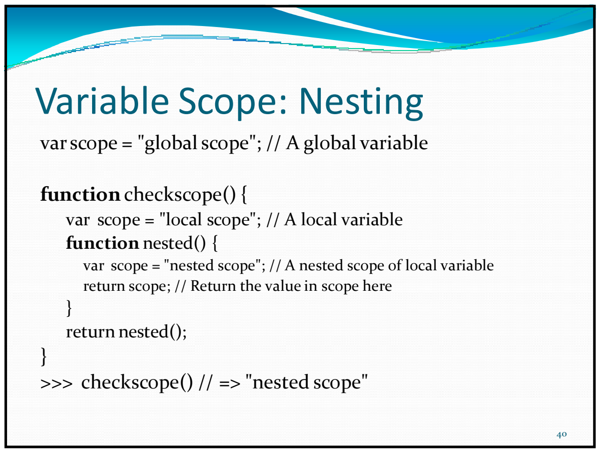
38



Variable Scope: Use of ***var***

```
scope = "global"; // Declare a global variable, even without var.

function checkscope2() {
    scope = "local"; // no var used:  Oops!  changed the global variable.
    myscope = "local"; // This implicitly declares a new global variable.
    return [scope, myscope]; // Return two values.
}
>>>  checkscope2() // => ["local", "local"]: has side effects!
>>>  scope // => "local": global variable has changed.
>>> myscope // => "local": global namespace cluttered up.
```

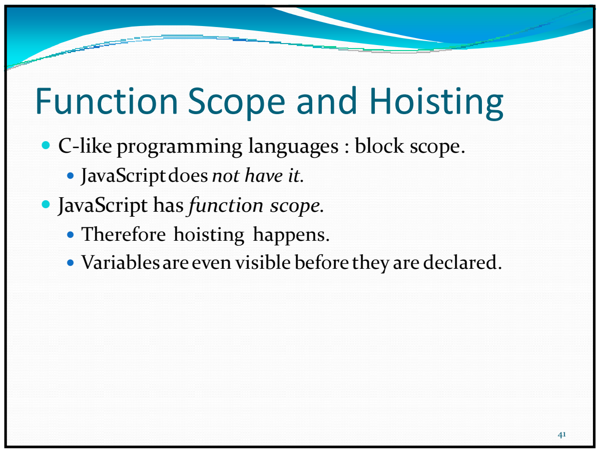
39



Variable Scope: Nesting

```
var scope = "global scope"; // A global variable

function checkscope() {
    var  scope = "local scope"; // A local variable
    function nested() {
        var  scope = "nested scope"; // A nested scope of local variable
        return scope; // Return the value in scope here
    }
    return nested();
}
>>>  checkscope() // => "nested scope"
```

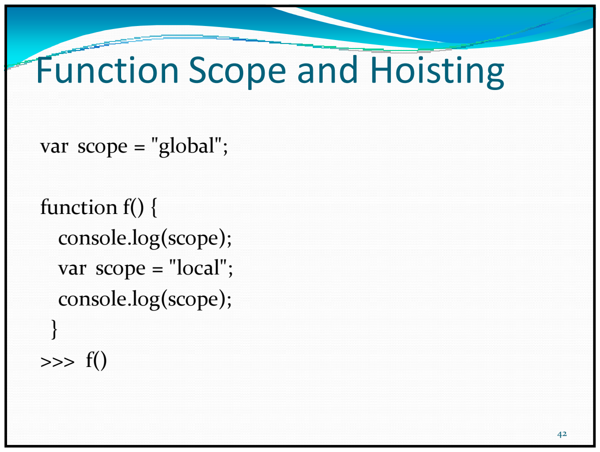
40



Function Scope and Hoisting

- C-like programming languages : block scope.
  - JavaScript does *not have it.*
- JavaScript has *function scope.*
  - Therefore hoisting happens.
  - Variables are even visible before they are declared.

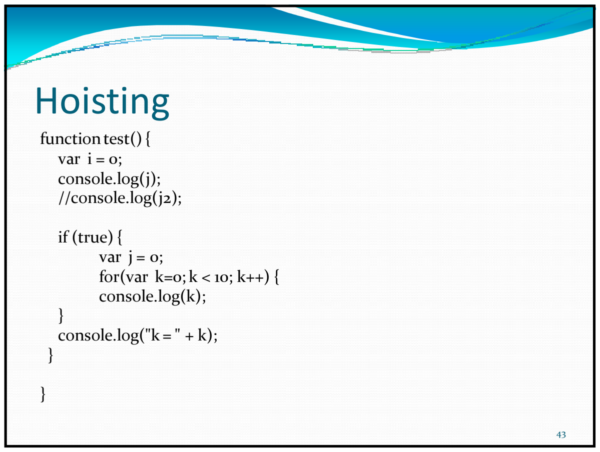
41



Function Scope and Hoisting

```
var  scope = "global";

function f() {
   console.log(scope);
   var  scope = "local";
   console.log(scope);
 }
>>>  f()
```

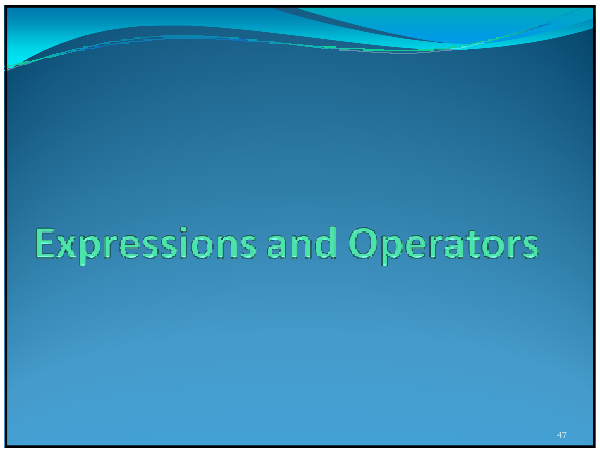
42



Hoisting

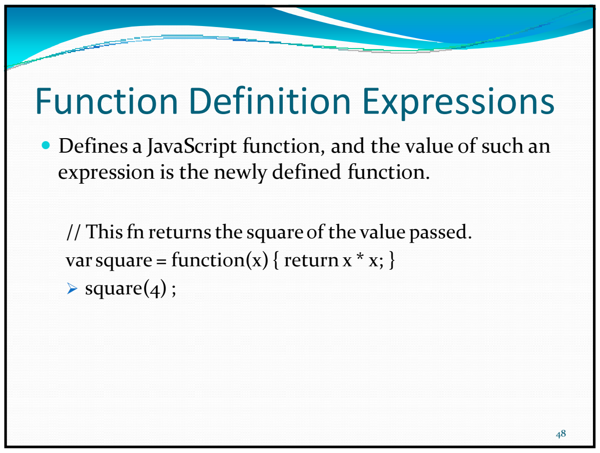
```
function test() {
   var  i = 0;
   console.log(j);
   //console.log(j2);

   if (true) {
          var  j = 0;
          for(var  k=0; k < 10; k++) {
          console.log(k);
   }
   console.log("k = " + k);
 }

}
```

43



Expressions and Operators

47

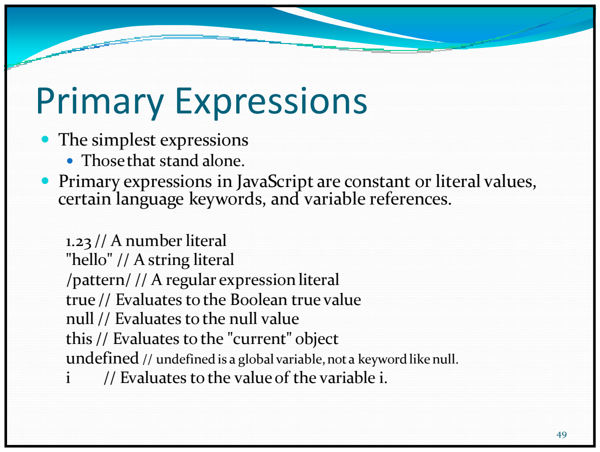


Function Definition Expressions

- Defines a JavaScript function, and the value of such an expression is the newly defined function.

```
// This fn returns the square of the value passed.
var square = function(x) { return x * x; }
➢ square(4) ;
```

48

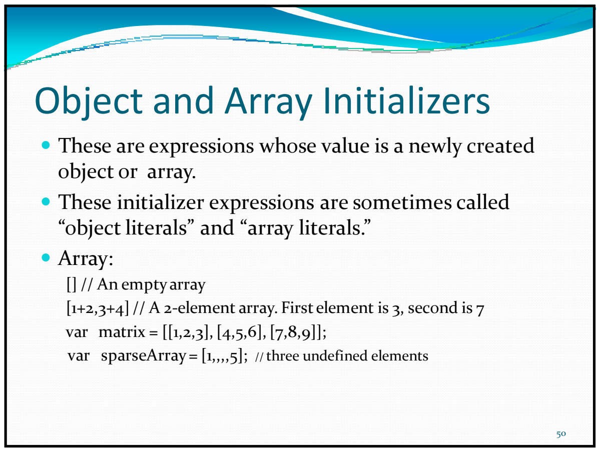


Primary Expressions

- The simplest expressions
  - Those that stand alone.
- Primary expressions in JavaScript are constant or literal values, certain language keywords, and variable references.

```
1.23 // A number literal
"hello" // A string literal
/pattern/ // A regular expression literal
true // Evaluates to the Boolean true value
null // Evaluates to the null value
this // Evaluates to the "current" object
undefined // undefined is a global variable, not a keyword like null.
i       // Evaluates to the value of the variable i.
```

49

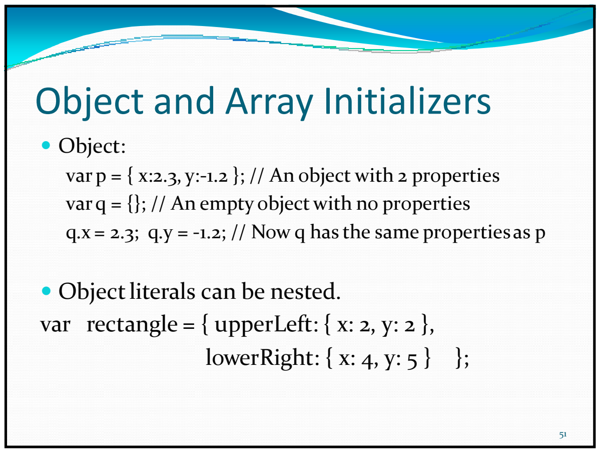


Object and Array Initializers

- These are expressions whose value is a newly created object or array.
- These initializer expressions are sometimes called “object literals” and “array literals.”
- Array:

```
[] // An empty array
[1+2,3+4] // A 2-element array. First element is 3, second is 7
var  matrix = [[1,2,3], [4,5,6], [7,8,9]];
var  sparseArray = [1,,,,5];  // three undefined elements
```

50



Object and Array Initializers

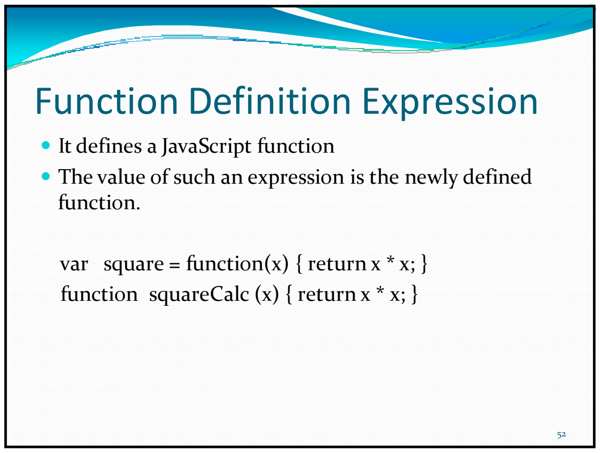
- Object:

```
var p = { x:2.3, y:-1.2 }; // An object with 2 properties
var q = {}; // An empty object with no properties
q.x = 2.3;  q.y = -1.2; // Now q has the same properties as p
```

- Object literals can be nested.

```
var   rectangle = { upperLeft: { x: 2, y: 2 },
                    lowerRight: { x: 4, y: 5 }     };
```

51

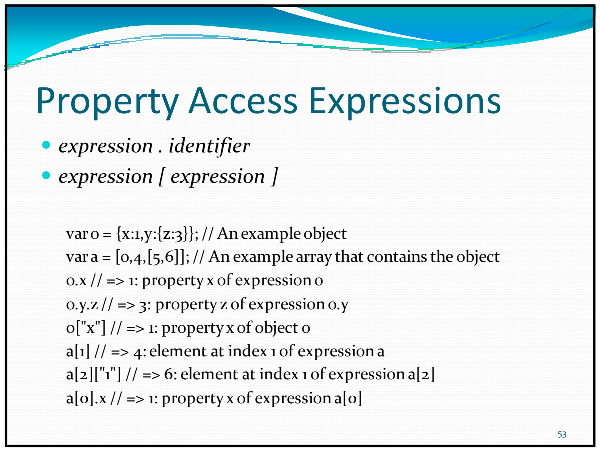


Function Definition Expression

- It defines a JavaScript function
- The value of such an expression is the newly defined function.

```
var   square = function(x) { return x * x; }
function  squareCalc (x) { return x * x; }
```

52

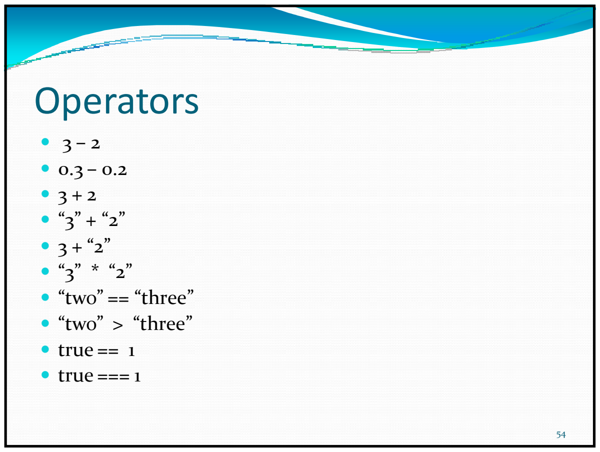


Property Access Expressions

- *expression . identifier*
- *expression [expression]*

```
var o = {x:1,y:{z:3}}; // An example object
var a = [0,4,[5,6]]; // An example array that contains the object
o.x // => 1: property x of expression o
o.y.z // => 3: property z of expression o.y
o["x"] // => 1: property x of object o
a[1] // => 4: element at index 1 of expression a
a[2]["1"] // => 6: element at index 1 of expression a[2]
a[0].x // => 1: property x of expression a[0]
```

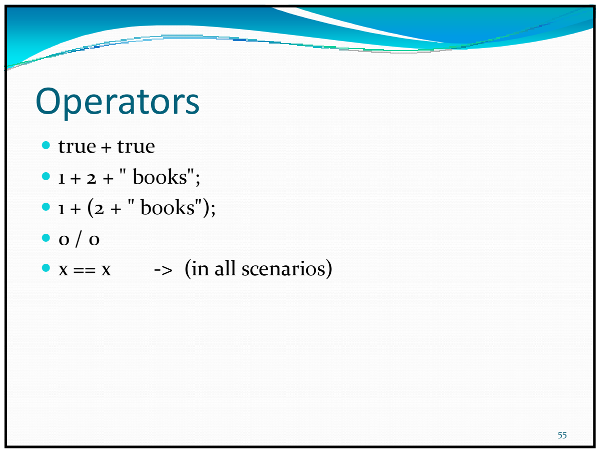
53



Operators

- 3 – 2
- 0.3 – 0.2
- 3 + 2
- “3” + “2”
- 3 + “2”
- “3” * “2”
- “two” == “three”
- “two” > “three”
- true == 1
- true === 1

54



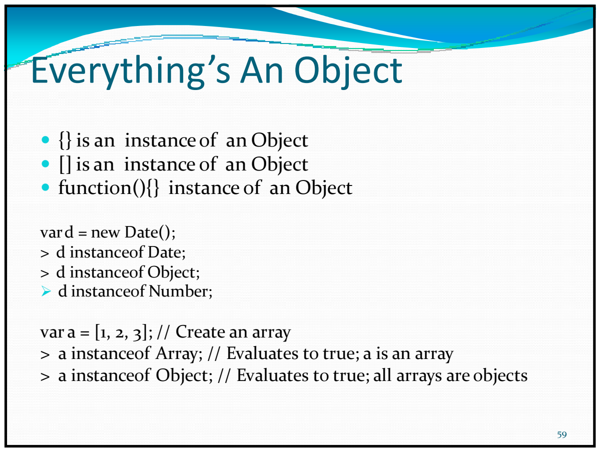
Operators

- true + true
- 1 + 2 + " books";
- 1 + (2 + " books");
- 0 / 0
- x == x -> (in all scenarios)

55

Objects

57

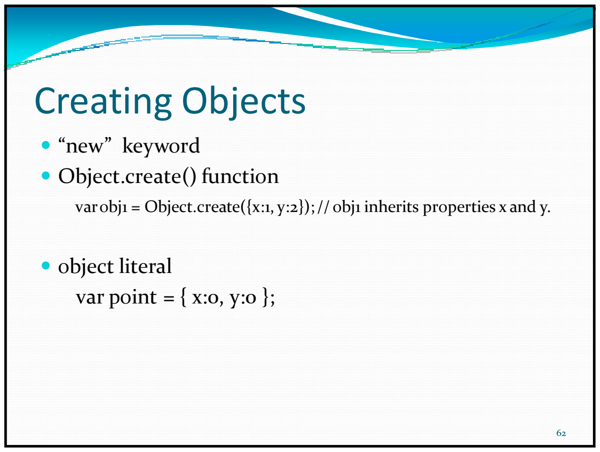


Everything's An Object

- {} is an instance of an Object
- [] is an instance of an Object
- function(){} instance of an Object

```
var d = new Date();
> d instanceof Date;
> d instanceof Object;
➢ d instanceof Number;

var a = [1, 2, 3]; // Create an array
> a instanceof Array; // Evaluates to true; a is an array
> a instanceof Object; // Evaluates to true; all arrays are objects
```

59



Creating Objects

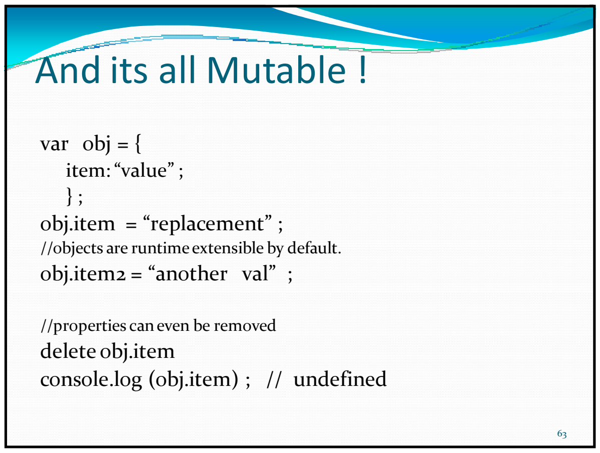
- "new" keyword
- Object.create() function

```
var obj1 = Object.create({x:1, y:2}); // obj1 inherits properties x and y.
```

- object literal

```
var point = { x:0, y:0 };
```

62



And its all Mutable !

```
var obj = {
    item: "value" ;
    } ;
obj.item = "replacement" ;
//objects are runtime extensible by default.
obj.item2 = "another val" ;

//properties can even be removed
delete obj.item
console.log (obj.item) ; // undefined
```

63