JavaScript - typing

- JavaScript is a *dynamically typed* programming language
  - variables are not defined by data type at declaration but by their values (or ‘literals’)
- The type of a literal is defined based on context (run-time)
- When combining literals of different types, the first type is used
- Java and C are *statically typed* – the type of the variable is set at compile time permanently

JavaScript - typeof

The `typeof` operator is unary – use of () optional

- e.g. `typeof`("pumpkin"), `typeof`(563), `typeof`(true), `typeof`(null), or `typeof` "squash"
- returns **type** of the operand: "`number`", "`string`", "`boolean`", "`object`", "`function`", `undefined`, "`xml`"

```
var a = "cherry";
var a_type = typeof(a);  // a_type is "string"
var b = 3.14;
var b_type = typeof(b);  // b_type is "number"
var c;
var c_type = typeof c;   // c_type is undefined
var d = null;
var d_type = typeof d;   // d_type is "object"
```

JavaScript – dynamic typing

```
var a = 99;
var b = "Ninety nine";
var c = 100 + 100;  // c is 200
var d = ( a < 100 );  // d is true
var e = d && (c > 100); // e is true
a = e;    // a is true
var f = “100” + 10; // f is 10010
var g = 100 – 10; // g is 90
```

JavaScript – weak typing

- JavaScript is also weakly typed
  - no restrictions on use of operators (such as the plus sign) involving values of different data types
- JavaScript rule: when you use + with a number and a string in any order you get a string result

```
var a = 100;
var b = "+100";
var sum = a + b;    // sum is "100+100" not 200
sum = a + parseInt(b);   // sum is 200
```

JavaScript - casting

- JavaScript data type examples
  - `"Count to " + 10` is `"Count to 10"`
  - and `2.5 + "10"` is `"2.510"`
- `parseInt()` and `parseFloat()` JavaScript functions cast values to a new type :
  - `parseInt( "12 dozen" )` returns the integer 12
  - `parseFloat( "33.23" ) returns  33.23`
  - `parseInt( "23.66") returns 23`
  - `parseInt("he is 40") returns NaN`
  - `parseInt("30 30 40") returns 30`
  - `parseInt( undefined )` and
  - `parseInt(null)` returns NaN (not a number)

```
<script type="text/javascript">

var answer = 99;

answer = "Ninety nine ";

var question = "What is 9 times 11? " +
answer;

document.write(question + "<br />");

question = answer + " is 9 times what
number?";

document.write(question);
</script>
```

Rock, Paper, Scissors

Work in groups to determine how you would write the 3 phases of the game:

a. User makes a choice
b. Computer makes a choice
c. A compare function will determine who wins

Let's look at one possible solution:
http://deepblue.cs.camosun.bc.ca/~cst000/comp140/test.html

JavaScript - expressions

- expressions in JavaScript come in four types
  - **assignment** which assigns a value to a variable
  - **arithmetic** evaluates to a number
  - **string** evaluates to a string
  - **logical** evaluates to a boolean value (true or false)
- use the keyword `var` to declare a variable and optionally assign it an initial value
- a variable declared using `var` with no initial value has the value ***undefined***
- it's possible to drop the `var` keyword but that makes the variable global scope -- not recommended

JavaScript - assignment

```
var x = 10;
var y = 5;

x += y;  // x is now 15 (10 + 5)

x *= y;  // x is now 75 (15 * 5)

x /= y;  // x is now 15 (75 / 5)

x %= y;  // x is now 0  (15 / 5 leaves 0
         //              remainder )
```

JavaScript - assignment

```
var x = 10;
var y = 5;
var z;

x++;  // increment operator; x is now 11

y--;  // decrement operator; y is now 4

z = ++y;  // z is 5 and y is now 5 (avoid
this)

z = x--;  // z is 11 and x is now 10 (avoid
too)
```

JavaScript - comparison

- use double equals sign (no space) == to test if two expressions are equivalent in value

  `1 == 1     "1" == 1     "100" == 99 + 1`
- use "bang equals" != for not-equal test

  `"a" != "A"    100 != 99.9    null != undefined`
- comparison operators < > <= >= test for less than, greater than, less than or equal, greater than or equal – these 3 are true:

  `100 < 111                3 < 100`

JavaScript - comparison

- triple equals sign === tests if two expressions are equivalent in **value** and the same ***type***

  1 === 1   "cat" === "cat"   1 === "1" (false)
- the !== tests if two expressions are not equivalent

  1 !== "1"   (but `1 == "1"` is `true`)
- null === undefined is false but
- null == undefined is true

JavaScript - logical

- logical AND operator is two ampersands: &&
- logical OR operator is two vertical pipes: ||
- logical NOT operator is a single bang: !

```
var x = 10;
var y = 5;

var a = ( x < y ) && ( x == 5 );  // false
var b = ( x > y ) || ( x < 5 );   // true

var c = !b;  // c is false
```

JavaScript – conditional

- ternary operator as in C, C++

  `(expression) ? value1 : value2;`

  - if (*expression*) evaluates true, then value1 is returned; otherwise, value2 is returned

```
var a = ( 3 == 4 ) ? "y" : "n";
 //  a is "n"
```

- can lead to cryptic programming code if overused

JavaScript – if block

- test a condition is met with an `if else` block

  if (*expression*) {
   block of statement(s) to execute if ***expression*** true

  } // do not forget the matching closing brace

```
var diff = 3-2;
if (diff == 1) {
    document.writeln("diff is 1");
}
```

JavaScript – if block

if-else statement version:

if (*expression*) {
   block of statements if *expression* true
} else {
   block of statements if *expression* false
}

```
    var diff = 3-2;
    if (diff == 1) {
        document.writeln("diff is 1");
    } else {
        document.writeln("diff is NOT 1");
    }
```

JavaScript - if block

```
var day = "Sunday";

if ( day == "Saturday" ) {
  document.writeln("It’s the
weekend!");
  the_weekend = true;
} else {
  document.writeln("Back to work.");
  the_weekend = false;
}
```

JavaScript – if block

- multiple tests combined into one if statement

```
var day = "Sunday";
var message;
if ( day == "Saturday" ) {
   message = "It’s the weekend!";
} else if ( day == "Monday" ) {
   message = " Back to work. ";
} else if ( day == "Friday" ) {
   message = " TGIF ! ";
} else {
   message = " Just another day. ";
}
```

JavaScript – if block

- when statement blocks are just one statement, the { } braces are optional

```
var day = "Sunday";
var message;
if ( day == "Saturday" )
   message = "It’s the start of the
weekend!";
else if ( day == "Monday" )
   message = " Back to work. ";
else if ( day == "Friday" )
   message = " TGIF ! ";
else
   message = "Just another day.";
```

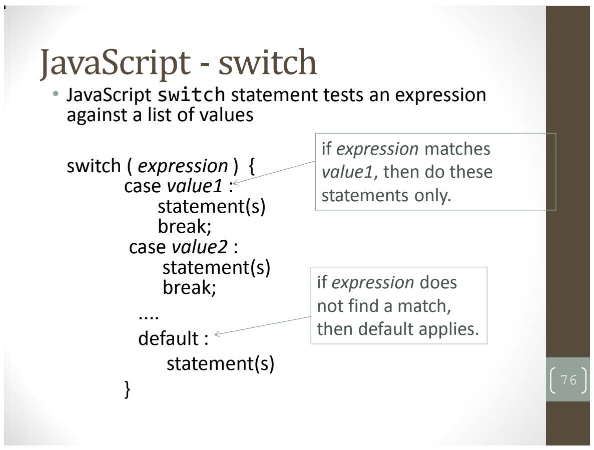
Nested if blocks

- it is possible to nest if statements within another if statement

```
var x = (2-3);
if (x < 0)
   sign = -1;
else {
  if (x == 0)
   sign = 0;
  else
   sign = 1;
}
// acceptable form
```

```
var x = (2-3);
if (x < 0)
   sign = -1;
if (x == 0)
   sign = 0;
if (x > 0)
   sign = 1;

// not recommended form
```



JavaScript - switch

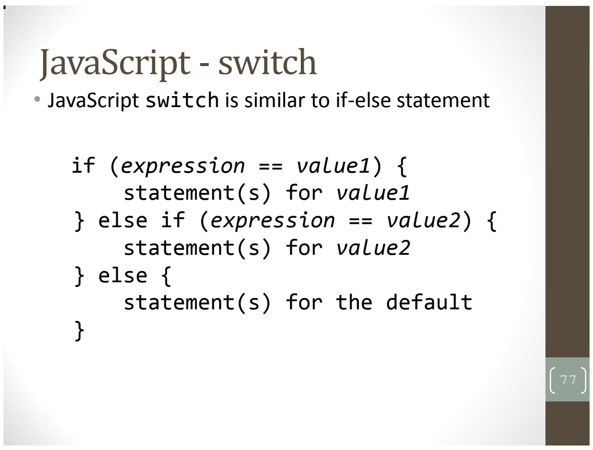
- JavaScript `switch` statement tests an expression against a list of values

```
switch ( expression ) {
     case value1 :
          statement(s)
          break;
     case value2 :
          statement(s)
          break;
     ....
     default :
          statement(s)
}
```

if *expression* matches *value1*, then do these statements only.

if *expression* does not find a match, then default applies.

76

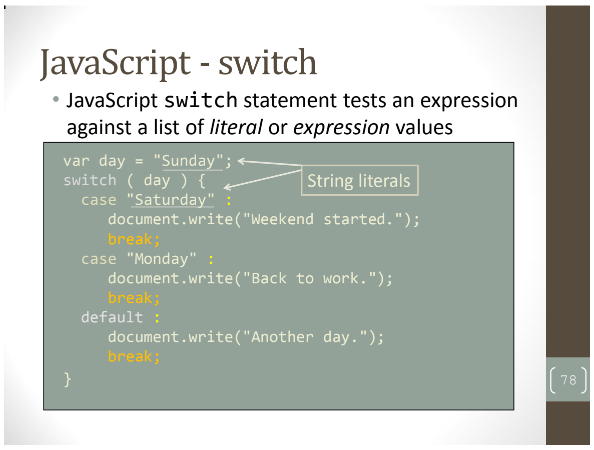


JavaScript - switch

- JavaScript `switch` is similar to if-else statement

```
if (expression == value1) {
    statement(s) for value1
} else if (expression == value2) {
    statement(s) for value2
} else {
    statement(s) for the default
}
```

77



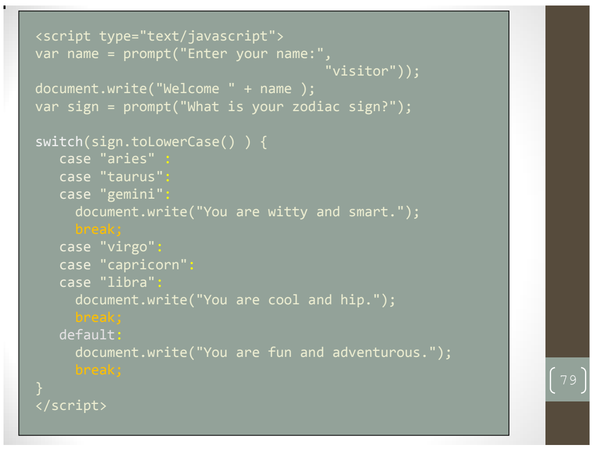
JavaScript - switch

- JavaScript `switch` statement tests an expression against a list of *literal* or *expression* values

```
var day = "Sunday";
switch ( day ) {
  case "Saturday" :
     document.write("Weekend started.");
     break;
  case "Monday" :
     document.write("Back to work.");
     break;
  default :
     document.write("Another day.");
     break;
}
```

String literals

78



```
<script type="text/javascript">
var name = prompt("Enter your name:",
                                    "visitor"));
document.write("Welcome " + name );
var sign = prompt("What is your zodiac sign?");

switch(sign.toLowerCase() ) {
   case "aries" :
   case "taurus":
   case "gemini":
     document.write("You are witty and smart.");
     break;
   case "virgo":
   case "capricorn":
   case "libra":
     document.write("You are cool and hip.");
     break;
   default:
     document.write("You are fun and adventurous.");
     break;
}
</script>
```

79

JavaScript - confirm

- `confirm` method allows the user to select an OK button or a Cancel button
- `confirm` returns true if OK clicked, false if Cancel clicked

```
if (confirm("Press OK to retry."))
  response = prompt("What is 2+2 ?", "3");
```

JavaScript - sample 2

- mathtest.html demonstrates the JavaScript `confirm` method in action

1 of 3

```
<script type="text/javascript">

  // define variables

  var question = "What is 10 + 10?";
  var answer = 20;
  var correct = '<img src="correct.gif">';
  var incorrect ='<img src="incorrect.gif">';
```

2 of 3

```
// ask the question

   var response = prompt(question,"0");

   // check the answer

   if (response != answer) {

     // wrong answer; retry once more.

     if (confirm("Wrong!  \
           Press OK for a second chance."))

       response = prompt(question, "0");
   }
```

3 of 3

```
// Check the answer.

  var output =
   (response == answer) ? correct : incorrect;

  // output will be one of these two strings:
       '<img src="correct.gif">'
       '<img src="incorrect.gif">'
</script>
   </head>
   <body>
   <script type="text/javascript">

   document.write(output);
   </script>
```

JavaScript – object literal

- a JavaScript object *literal* is delimited by { } which contains the object's ***properties*** as name:value pairs, separated by commas.

```
var student = { name: "Smith, John",
                id: 103923,
                program: "CSC",
                dob: new Date(1990, 3, 20)
  };

document.write( student.name ); // Smith, John
document.write( student["program"]); //  CSC
```

JavaScript - eval

- `eval()` method
  - evaluates a string parameter to its numeric value
    - e.g. eval("4 + 5") returns a value of 9
  - avoid using eval if possible – there are potential side effects, especially if the string parameter contains malicious code
- http://javascriptweblog.wordpress.com/2010/04/19/how-evil-is-eval/

JavaScript - iteration

- iteration is the process of repeating the execution of one or more statements until some end condition is reached
  - each time the iteration body is executed is a ***cycle***
- example 1 : continually prompt user until right answer is entered
- example 2 : display the month names (January, February, etc) of the entire year
- example 3 : calculate and show the values of a multiplication table up to 12 x 12

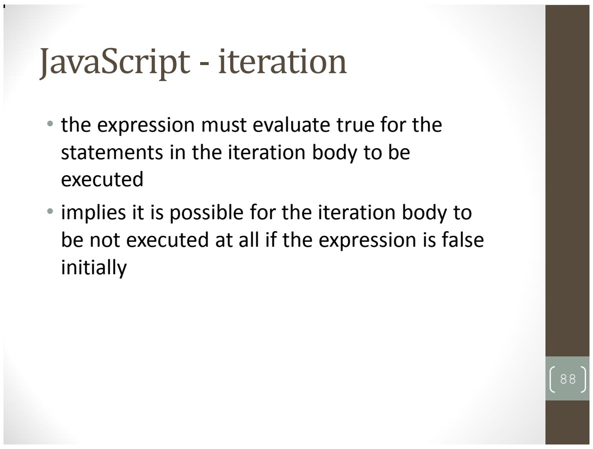
JavaScript - iteration

- the while statement indicates iteration
- conceptually:

```
while ( condition is true )
    perform these statement(s) within
        body of iteration in order continually
```

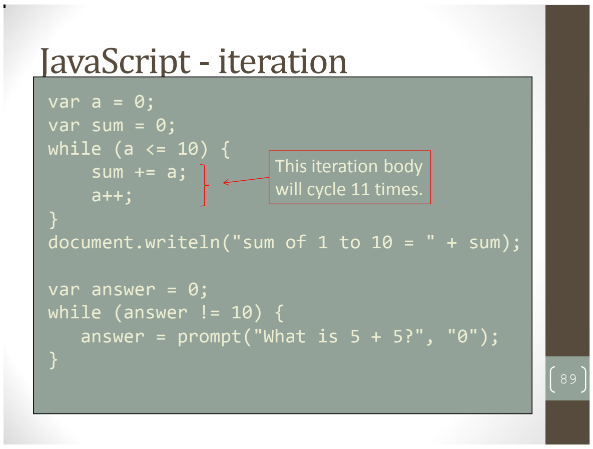
- in practice:

```
while ( expression ) {
    one or more statements;
}
```



JavaScript - iteration

- the expression must evaluate true for the statements in the iteration body to be executed
- implies it is possible for the iteration body to be not executed at all if the expression is false initially

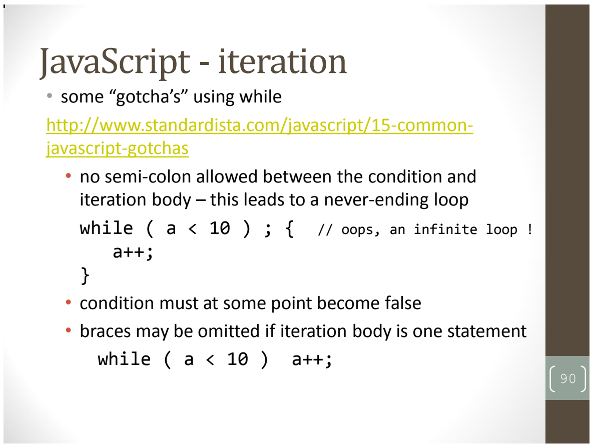


JavaScript - iteration

```
var a = 0;
var sum = 0;
while (a <= 10) {
    sum += a;
    a++;
}
document.writeln("sum of 1 to 10 = " + sum);

var answer = 0;
while (answer != 10) {
   answer = prompt("What is 5 + 5?", "0");
}
```

This iteration body will cycle 11 times.



JavaScript - iteration

- some "gotcha's" using while

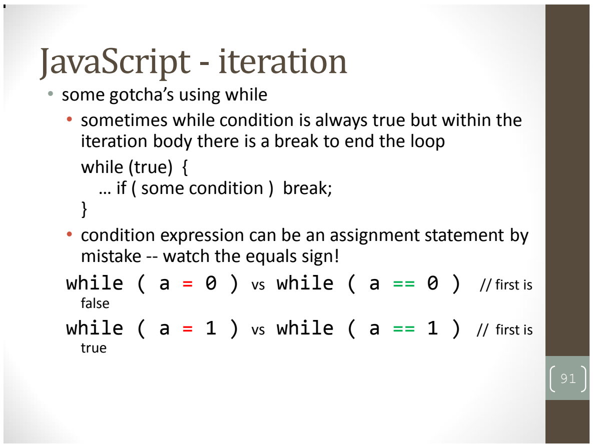
http://www.standardista.com/javascript/15-common-javascript-gotchas

  - no semi-colon allowed between the condition and iteration body – this leads to a never-ending loop

```
while ( a < 10 ) ; {  // oops, an infinite loop !
   a++;
}
```

  - condition must at some point become false
  - braces may be omitted if iteration body is one statement

```
while ( a < 10 )  a++;
```



JavaScript - iteration

- some gotcha's using while
  - sometimes while condition is always true but within the iteration body there is a break to end the loop

```
while (true) {
   … if ( some condition )  break;
}
```

  - condition expression can be an assignment statement by mistake -- watch the equals sign!

```
while ( a = 0 ) vs while ( a == 0 )  // first is false
while ( a = 1 ) vs while ( a == 1 )  // first is true
```

JavaScript - iteration

- Some gotcha's using while
- forgetting to increment the counter if it is used in the condition

```
 var n = 0;
 var sum = 0;
 while ( n < 10 ) {
     sum += n; }
}
// oops, n is always zero!!
```

JavaScript - iteration

- Another form of iteration: for
- Useful when number of iterations is known
- Conceptually:

```
for ( x in array){
elem=array[x];
        elem.method(s);}
```

- In practice:

```
for ( optional initial statement(s);
    condition;
    optional end body statement(s) )
        execute statement(s)
```

JavaScript - iteration

```
var sum = 0;
for (var n = 0; n <= 10; n++)  {
    sum += n;
}
document.writeln("sum of 1 to 10 = " + sum);


var pets = new Array( "cat", "dog", "fish" );
for (var i=0, len=pets.length; i < len; i++) {
    document.write("I have a "
                  + pets[i]
                  + ". <br />");
}
```

JavaScript - iteration

- If only one statement in body, braces may be omitted

```
for (var n = 0; n < 10; n++)  sum
+= n;
```

- The initial statement and end statement (usually an increment) are optional

```
var n = 0;
for (  ;  n < 10 ; )  {
    … n++;
   }
```

- same as a while loop

JavaScript - iteration

- If the condition is false initially, the iteration body will not be executed at all and execution will proceed with the next statement after the end of the iteration body

```
var sum = 0;
for (var n = 0; n > 1; n-- ) {   //
  oops, 0 should be 10
  sum += n;        // never executed
}
document.write( "sum is " + sum );
  // sum is 0
```

JavaScript - iteration

- The do while iteration is similar to while but the condition is after the iteration body
- Guarantees the iteration body is executed at least once

```
var  n = 0;
do  {
        n++;
         document.write("n has value "
                        + n );
}  while ( n < 10 ) ;
```

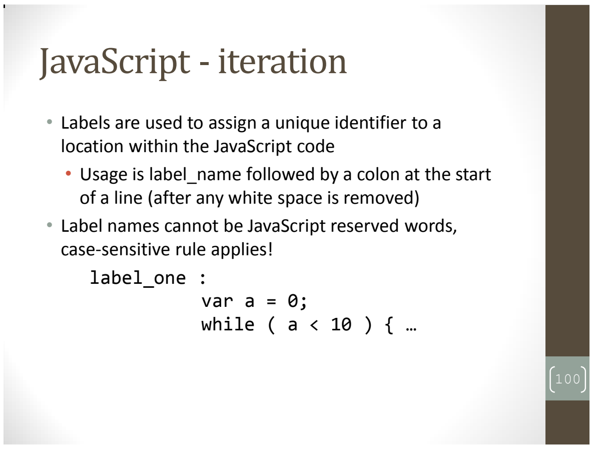
JavaScript - iteration

- An iteration body may include an iteration
- "outer loop" contains an "inner loop"

```
 var a = 0;
while ( a < 10) {
     var b = 0;
     while ( b < 10) {
            document.writeln( a * b );
            b++;
      }
       document.writeln( "< /br>" );
       a++;
}
```

JavaScript - iteration

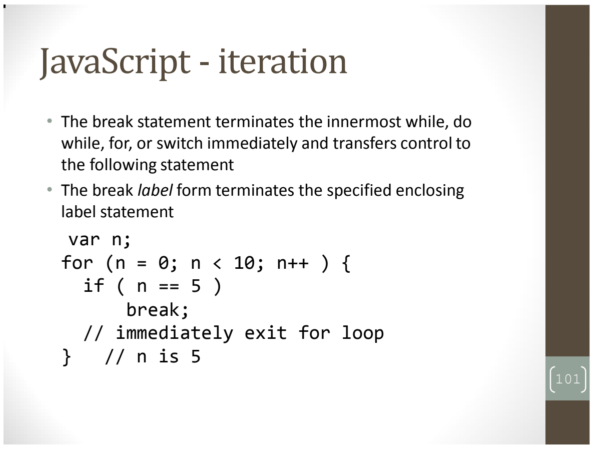
```
 for (var a = 0;  a < 10; a++ ) {
  for ( var b = 0; b < 10; b++ )
     document.writeln( a * b );
   document.writeln("<br />");
}
```



JavaScript - iteration

- Labels are used to assign a unique identifier to a location within the JavaScript code
  - Usage is label_name followed by a colon at the start of a line (after any white space is removed)
- Label names cannot be JavaScript reserved words, case-sensitive rule applies!

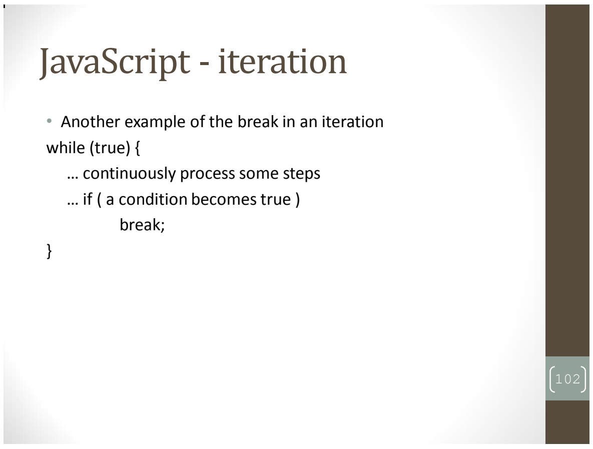
```
label_one :
          var a = 0;
          while ( a < 10 ) { …
```



JavaScript - iteration

- The break statement terminates the innermost while, do while, for, or switch immediately and transfers control to the following statement
- The break *label* form terminates the specified enclosing label statement

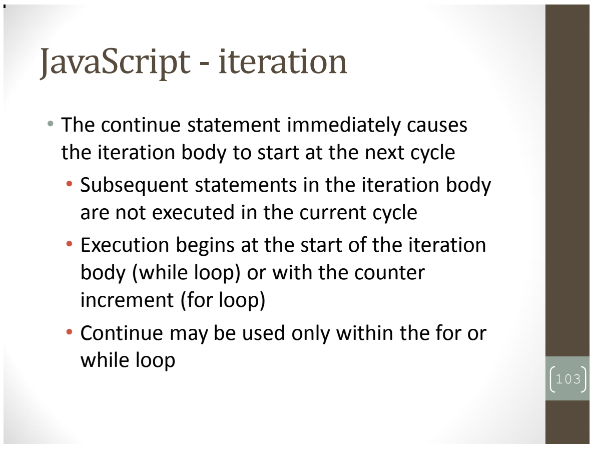
```
 var n;
for (n = 0; n < 10; n++ ) {
  if ( n == 5 )
      break;
  // immediately exit for loop
}   // n is 5
```



JavaScript - iteration

- Another example of the break in an iteration

```
while (true) {
   … continuously process some steps
   … if ( a condition becomes true )
           break;
}
```



JavaScript - iteration

- The continue statement immediately causes the iteration body to start at the next cycle
  - Subsequent statements in the iteration body are not executed in the current cycle
  - Execution begins at the start of the iteration body (while loop) or with the counter increment (for loop)
  - Continue may be used only within the for or while loop

JavaScript - iteration

```
// Sum up the odd integers from 0 to 20.
var sum = 0;

for ( var a=0; a <= 20; a++) {

    if ( a % 2 == 0 ) {
        continue;
    }
    sum += a;
}
```

JavaScript - iteration

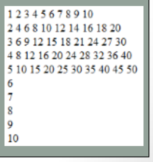
- Break and continue may indicate an optional label, e.g.
  `break calculateSum;`
  `continue releaseMemory;`
- break *label* means stop executing the statement at label (likely a loop of some kind)
- continue *label* means transfer execution to the statement at label

JavaScript - iteration

```
//Outer:
 for ( var a=1; a <= 10; a++ ) {

//Inner:
   for ( var b=1; b <= 10; b++ ) {

     document.write( (a*b) + "  " );
     if ( a > 5 ) {
        break Inner;
     }
   }
   document.write( "<br />");
}
```

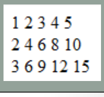


```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6
7
8
9
10
```

JavaScript - iteration

```
//Outer:
 for ( var a=1; a <= 5; a++ ) {

//Inner:
   for ( var b=1; b <= 5; b++ ) {
     if ( a > 5 ) {
        continue Inner;
     }
     document.write( (a*b) + "  " );
   }
   document.write( "<br />");
}
```



```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
```

JavaScript - iteration

- Use the while iteration when you do not know in advance the number of iterations
- Use the for iteration when you do know in advance the number of iterations
- Avoid use of break and continue if possible
  - Misuse or overuse can lead to ‘code spaghetti’