

Introduction to Javascript

Module 2



Sample code 1: Displaying a line of text

```
<html>
<head>
  <title>A First Program in JavaScript</title>
<script type = "text/javascript">
  document.writeln(
    "<h1>Welcome to JavaScript Programming!</h1>" );
  </script>
</head><body></body>
</html>
```

Sample code I

- In HTML, JavaScript code is inserted between `<script>` and `</script>` tags.
- `<script>` tag: indicate to the browser that the text which follows is part of a script
- **type** attribute : the type of file as well as the **scripting language** used in the script—in this case, a text file written in javascript (not required)
- Scripts can be placed in the `<body>`, or in the `<head>` section of an HTML page, or in both
- String: also called as character string/message/string literal

Sample code I

- `document.writeln("<h1>Welcome to JavaScript Programming!</h1>");`
- Above line of code is a statement
- use the browser's **document object** to specify text to display in the HTML document.
- Method **writeln** instructs the browser to display the argument string

Code 2: Printing one line with separate statements

```
✓ <html>
✓ <head>
  <title>A First Program in JavaScript</title>
✓ <script type = "text/javascript">
  document.write( "<h1 style = \"color: magenta\">" );
✓ document.write( "Welcome to JavaScript " +
  "Programming!</h1>" );
  </script>
</head><body></body>
</html>
```

Code 2

- `writeln()` adds a newline character after each statement. `write()` does not.
- “+” operator (concatenation operator) : joins two strings together.
- The **backslash** (`\`) in a string is an **escape character**.
- When a backslash is encountered in a string of characters, the next character is combined with the backslash to form an **escape sequence**
- `document.write("<h1 style = \"color: magenta\">");`
- `document.write("<h1 style = 'color: magenta'>");`
- Above syntax is also used.

Code 3: write() and writeln()

```
<!DOCTYPE html>
<html>
<body>
<h1>The Document Object</h1>
<h2>The write() and writeln() Methods</h2>
<pre>
<script>
document.write("Hello World!");
document.write("Have a nice day!");
document.write("<br>");
document.writeln("Hello World!");
document.writeln("Have a nice day!");
</script>
</pre>
</body>
</html>
```

The Document Object

The write() and writeln() Methods

Hello World!Have a nice day!
Hello World!
Have a nice day!

Code 3

- The `\n`, `\t` and `\r` escape sequences in the table do not affect XHTML rendering unless they are in a **pre element**
- **pre element** displays the text between its tags in a fixed-width font exactly as it is formatted between the tags, including leading white-space characters and consecutive white-space characters

Code4: Printing on multiple lines with a single statement

```
1  <html>
2  <head>
3  <title>A First Program in JavaScript</title>
4  <script type = "text/javascript">
5      document.write( "<h1 style = 'color: magenta'>" );
6      //document.write( "<h1 style = \"color: magenta\">" );
7      document.writeln( "<h1>Welcome to<br />JavaScript" + "<br />Programming!</h1>" );
8      </script>
9  </head><body></body>
10 </html>
```

Code5: Alert dialog displaying multiple lines

```
1 <html>
2 <head>
3   <title>A First Program in JavaScript</title>
4 <script type = "text/javascript">
5   window.alert( "Welcome to\nJavaScript\nProgramming!" );
6 </script>
7 </head>
8 <body>
9   <p>Click Refresh (or Reload) to run this script again.</p>
10 </body>
11 </html>
```

Code 5

- Above script uses the browser's **window** object to display an alert dialog
- The argument to the **window** object's **alert** method is the string to display
- *HTML elements in an alert dialog's message are not interpreted as HTML.*
- *This means that using `
`, for example, to create a line break in an alert box is an error.*
- *The string `
` will simply be included in your message.*

Common escape sequences

Escape sequence	Description
<code>\n</code>	New line. Position the screen cursor at the beginning of the next line.
<code>\t</code>	Horizontal tab. Move the screen cursor to the next tab stop.
<code>\r</code>	Carriage return. Position the screen cursor to the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
<code>\\</code>	Backslash. Used to represent a backslash character in a string.
<code>\"</code>	Double quote. Used to represent a double-quote character in a string contained in double quotes. For example, <pre>window.alert("\"in quotes\"");</pre> displays "in quotes" in an alert dialog.
<code>\'</code>	Single quote. Used to represent a single-quote character in a string. For example, <pre>window.alert('\''in quotes\'');</pre> displays 'in quotes' in an alert dialog.



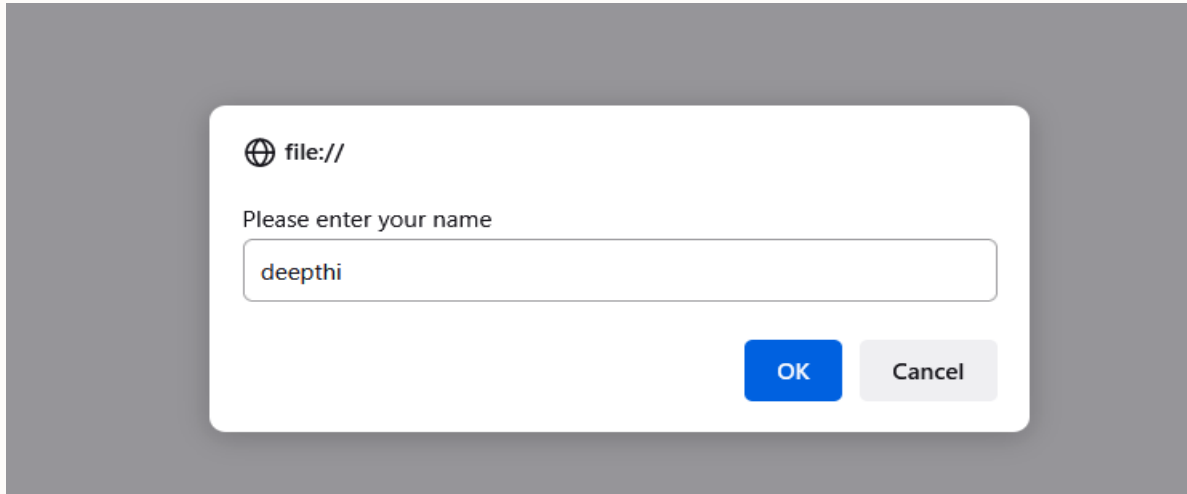
Obtaining user inputs with prompt dialogs

- A script can adapt the content based on input from the user or other variables, such as the time of day or the type of browser used by the client
- Dynamic web pages

Code6: Dynamic welcome page

```
1  <html>
2  <head>
3  <title>Using Prompt and Alert Boxes</title>
4  <script type = "text/javascript">
5      var name;
6      name = window.prompt( "Please enter your name" );
7      document.writeln( "<h1>Hello, " + name +", welcome to JavaScript programming!</h1>" );
8  </script>
9  </head>
10 <body>
11     <p>Click Refresh (or Reload) to run this script again.</p>
12 </body>
13 </html>
```

Code6: Dynamic welcome page

A browser dialog box with a white background and rounded corners, set against a dark gray background. At the top left, there is a globe icon followed by the text "file://". Below this, the text "Please enter your name" is displayed. Underneath the text is a text input field containing the name "deepthi". At the bottom right of the dialog, there are two buttons: a blue "OK" button and a gray "Cancel" button.

file://

Please enter your name

deepthi

OK Cancel

Hello, deepthi, welcome to JavaScript programming!

Click Refresh (or Reload) to run this script again.

Code6: Dynamic welcome page

- The keyword **var** at the beginning of the statement indicates that the word name is a **variable**
- **var name; ? declaration**
- The name of a variable can be any valid **identifier**.
- An identifier is a series of characters consisting of letters, digits, underscores (_) and dollar signs (\$) that does not begin with a digit and is not a reserved JavaScript keyword
- Identifiers may not contain spaces
- Eg: Welcome, \$value, _value, m_inputField1 and button7
- Input field, 7button : invalid identifiers
- JavaScript is **case sensitive**—uppercase and lowercase letters are considered to be different characters, so **name**, **Name** and **NAME** are different identifiers.

Code6: Dynamic welcome page

- By convention, variable-name identifiers begin with a lowercase first letter.
- Each subsequent word should begin with a capital first letter.
- For example, identifier `itemPrice` has a capital P in its second word, Price
- **comma-separated list** of variable names : Declaration can be split over several lines with each variable in the declaration separated by a comma
- Line 6 calls the window object's `prompt` method, which displays the dialog
- The argument to `prompt` specifies a message telling the user what to type in the text field.
- An optional second argument, separated from the first by a comma, may specify the default string displayed in the text field; our code does not supply a second argument

Code6: Dynamic welcome page

- If the user clicks the **Cancel** button, no string value is sent to the program.
- Instead, the prompt dialog submits the value **null**, a JavaScript keyword signifying that a variable has no value.
- Line 6 **assigns** the value returned by the window object's prompt method (a string containing the characters typed by the user—or the default value or null if the **Cancel** button is clicked) to variable name by using the **assignment operator**, =.
- This entire statement is called an **assignment statement** because it assigns a value to a variable.
- The expression to the right of the assignment operator is always evaluated first.

Code6: Dynamic welcome page

- HTML page is not rendered until the prompt is dismissed because the prompt pauses execution in the head, before the body is processed.

Code 7: Arithmetic: Adding integers

```
1  <html>
2    <head>
3      <title>An Addition Program</title>
4      <script type = "text/javascript">
5        var firstNumber; // first string entered by user
6        var secondNumber; // second string entered by user
7        var number1; // first number to add
8        var number2; // second number to add
9        var sum; // sum of number1 and number2
10       firstNumber = window.prompt( "Enter first integer" );
11       secondNumber = window.prompt( "Enter second integer" );
12       number1 = parseInt( firstNumber );
13       number2 = parseInt( secondNumber );
14       sum = number1 + number2;
15       document.writeln( "<h1>The sum is " + sum + "</h1>" );
16     </script>
17   </head>
18   <body>
19     <p>Click Refresh (or Reload) to run the script again</p>
20   </body>
21 </html>
```

Code 7: Adding integers

- In the above example, if the user either types a noninteger value or clicks the **Cancel** button, a logic error will occur, and the sum of the two values will appear in the XHTML document as **NaN** (meaning **not a number**).
- Line 12-13: convert the two strings input by the user to integer values that can be used in a calculation.
- Function **parseInt** converts its string argument to an integer.

Arithmetic

- There is no arithmetic operator for exponentiation in JavaScript.
- Arithmetic expressions in JavaScript must be written in **straight-line form** to facilitate entering programs into the computer.
- Expressions such as “a divided by b” must be written as a / b , so that all constants, variables and operators appear in a straight line

Arithmetic operators

JavaScript operation	Arithmetic operator	Algebraic expression	JavaScript expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	bm	<code>b * m</code>
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
Remainder	%	$r \bmod s$	<code>r % s</code>

Arithmetic: rules of operator precedence

- Multiplication, division and remainder operations are applied first.
- If an expression contains several multiplication, division and remainder operations, operators are applied from left to right.
- Multiplication, division and remainder operations are said to have the same level of precedence.
- Addition and subtraction operations are applied next.
- When we say that operators are applied from left to right, we are referring to the **associativity** of the operators—the order in which operators of equal priority are evaluated.

. Arithmetic: rules of operator precedence

Operator(s)	Operation(s)	Order of evaluation (precedence)
*, / or %	Multiplication Division Remainder	Evaluated first. If there are several such operations, they are evaluated from left to right.
+ or -	Addition Subtraction	Evaluated last. If there are several such operations, they are evaluated from left to right.

Algebraic and equivalent javascript expr

Algebra: $m = \frac{a + b + c + d + e}{5}$

JavaScript: `m = (a + b + c + d + e) / 5;`

Algebra: $y = mx + b$

JavaScript: `y = m * x + b;`

Algebra: $z = pr \% q + w / x - y$

Java: `z = p * r % q + w / x - y;`



`y = a * x * x + b * x + c;`



Algebraic and equivalent javascript expr

- As in algebra, it is acceptable to use unnecessary parentheses in an expression to make the expression clearer. These are also called **redundant parentheses**.
- $y = (a * x * x) + (b * x) + c;$

Decision making: equality and relational operators

Standard algebraic equality operator or relational operator	JavaScript equality or relational operator	Sample JavaScript condition	Meaning of JavaScript condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Decision making: equality and relational operators

- Equality operator have lower precedence than relational operator
- The equality operators also associate from left to right.

Code 8: Using equality and relational operators

```
1  <html>
2  <head>
3  <title>Using Relational Operators</title>
4  <script type = "text/javascript">
5  var name;
6  var now = new Date();// current date and time
7  var hour = now.getHours();// current hour (0-23)
8  // read the name from the prompt box as a string
9  name = window.prompt( "Please enter your name" );
10 if ( hour < 12 )
11 document.write( "<h1>Good Morning, " );
12 if ( hour >= 12 )
13 {
14 hour = hour - 12;
15 if ( hour < 6 )
16 document.write( "<h1>Good Afternoon, " );
17 if ( hour >= 6 )
18 document.write( "<h1>Good Evening, " );
19 } // end if
20 document.writeln( name +
21 ", welcome to JavaScript programming!</h1>" );
22 </script>
23 </head>
24 <body>
25 <p>Click Refresh (or Reload) to run this script again.</p>
26 </body>
27 </html>
```

Using equality and relational operators

- We use JavaScript's built-in Date object to acquire the current local time.
- We create a new instance of an object by using the **new** operator followed by the type of the object, Date, and a pair of parentheses.
- In line 7, variable hour is set to an integer equal to the current hour (in a 24-hour clock format) returned by the Date object's getHours method.

Control statements

- **if** statement is called a **single-selection structure** because it selects or ignores a single action
- **if...else** statement is a **double-selection structure** because it selects between two different actions
- The **switch** statement is a **multiple-selection structure** because it selects among many different actions
- Repetition structures: **while**, **do-while**, **for**, **for.....in**
- JavaScript has only eight control structures: sequence, three types of selection and four types of repetition

JavaScript keywords

JavaScript keywords

<code>break</code>	<code>case</code>	<code>catch</code>	<code>continue</code>	<code>default</code>
<code>delete</code>	<code>do</code>	<code>else</code>	<code>false</code>	<code>finally</code>
<code>for</code>	<code>function</code>	<code>if</code>	<code>in</code>	<code>instanceof</code>
<code>new</code>	<code>null</code>	<code>return</code>	<code>switch</code>	<code>this</code>
<code>throw</code>	<code>true</code>	<code>try</code>	<code>typeof</code>	<code>var</code>
<code>void</code>	<code>while</code>	<code>with</code>		

Keywords that are reserved but not used by JavaScript

<code>abstract</code>	<code>boolean</code>	<code>byte</code>	<code>char</code>	<code>class</code>
<code>const</code>	<code>debugger</code>	<code>double</code>	<code>enum</code>	<code>export</code>
<code>extends</code>	<code>final</code>	<code>float</code>	<code>goto</code>	<code>implements</code>
<code>import</code>	<code>int</code>	<code>interface</code>	<code>long</code>	<code>native</code>
<code>package</code>	<code>private</code>	<code>protected</code>	<code>public</code>	<code>short</code>
<code>static</code>	<code>super</code>	<code>synchronized</code>	<code>throws</code>	<code>transient</code>
<code>volatile</code>				

Control statements

- **Single-entry/single-exit control structures** make it easy to build programs
- The control structures are attached to one another by connecting the exit point of one to the entry point of the next. This process is similar to the way in which a child stacks building blocks, so we call it **control-structure stacking**.

Control statements

- if selection statement
- if statement is a single-entry/single-exit control structure.
- Flowcharts for control structures contain (besides small circle symbols and flowlines) only rectangle symbols, to indicate the actions to be performed, and diamond symbols, to indicate decisions to be made. This type of flowchart represents the **action/decision model of programming**.

Control statements

- If---else statement
- Syntax
- `if (condition) {`
 // block of code to be executed if the condition is true
`} else {`
 // block of code to be executed if the condition is false
`}`

Code9: if---else statement

```
1. <!DOCTYPE html>
2. <html>
3. <body>
4. <h2>JavaScript if .. else</h2>
5. <p>A time-based greeting:</p>
6. <p id="demo"></p>
7. <script>
8.   const hour = new Date().getHours();
9.   let greeting;
10.  if (hour < 18) {
11.    greeting = "Good day";
12.  } else {
13.    greeting = "Good evening";
14.  }
15.  document.getElementById("demo").innerHTML = greeting;
16. </script>
17. </body>
18. </html>
```

If-else

- let allows you to declare variables that are limited to the scope of a block statement, or expression on which it is used.
- unlike the var keyword, which declares a variable globally, or locally to an entire function regardless of block scope
- The getElementById() is a **DOM method used to return the element that has the ID attribute with the specified value**
- The Element property innerHTML gets or sets the HTML or XML markup contained within the element

Code 10: Conditional-operator

```
1. <!DOCTYPE html>
2. <html>
3. <body>
4. <h1>JavaScript Comparison</h1>
5. <h2>The () ? : Ternary Operator</h2>
6. <p>Input your age and click the button:</p>
7. <input id="age" value="18" />
8. <button onclick="myFunction()">Try it</button>
9. <p id="demo"></p>
10. <script>
11. function myFunction() {
12.   let age = document.getElementById("age").value;
13.   let voteable = (age < 18) ? "Too young":"Old enough";
14.   document.getElementById("demo").innerHTML = voteable + " to vote.";
15. }
16. </script>
17. </body>
18. </html>
```

JavaScript Comparison

The () ? : Ternary Operator

Input your age and click the button:

 Try it

Too young to vote.

Code 10: Conditional-operator

- The <button> tag defines a clickable button
- The onclick attribute fires on a mouse click on the element
- The id attribute specifies a unique id for an HTML element
- The id attribute is most used to point to a style in a style sheet, and by JavaScript to manipulate the element with the specific id

Code 11: Nested if-else

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <h2>JavaScript if .. else</h2>
5  <p>A time-based greeting:</p>
6  <p id="demo"></p>
7  <script>
8  const time = new Date().getHours();
9  let greeting;
10 if (time < 10) {
11   greeting = "Good morning";
12 } else if (time < 20) {
13   greeting = "Good day";
14 } else {
15   greeting = "Good evening";
16 }
17 document.getElementById("demo").innerHTML = greeting;
18 </script>
19 </body>
20 </html>
```

Nested if-else

- **Dangling-else problem**
- JavaScript interpreter always associates an else with the previous if, unless told to do otherwise by the placement of braces ({}). This situation is referred to as the **dangling-else problem**

```
if ( x > 5 )
    if ( y > 5 )
        document.writeln( "x and y are > 5" );
else
    document.writeln( "x is <= 5" );
```

```
if ( x > 5 )
{
    if ( y > 5 )
        document.writeln( "x and y are > 5" );
}
else
    document.writeln( "x is <= 5" );
```

A set of statements contained within a pair of braces is called a block

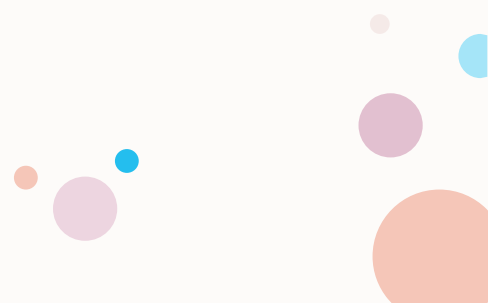
Control structures

- Syntax errors (e.g., when one brace in a block is left out of the program) are caught by the interpreter when it attempts to interpret the code containing the syntax error.
- A **logic error** (e.g., the one caused when both braces around a block are left out of the program) also has its effect at execution time.
- A **fatal logic error** causes a program to fail and terminate prematurely.
- A **nonfatal logic error** allows a program to continue executing, but the program produces incorrect results.



• While statement

- A repetition structure (also known as a **loop**) allows the programmer to specify that a script is to repeat an action while some condition remains true.



Code 12; counter controlled repetition

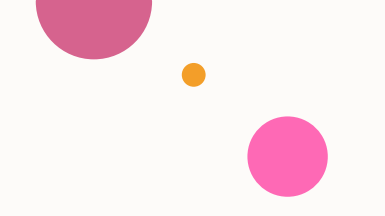

```
1  <html>
2  <head>
3    <title>Class Average Program</title>
4  <script type = "text/javascript">
5    var total; // sum of grades
6    var gradeCounter; // number of grades entered
7    var grade; // grade typed by user (as a string)
8    var gradeValue; // grade value (converted to integer)
9    var average; // average of all grades
10   // Initialization Phase
11   total = 0; // clear total
12   gradeCounter = 1;
13   // prepare to loop
14   // Processing Phase
15   // loop 10 times
16  while ( gradeCounter <= 5 ){
17    // prompt for input and read grade from user
18    grade = window.prompt( "Enter integer grade:", "0" );
19    // convert grade from a string to an integer
20    gradeValue = parseInt( grade );
21    // add gradeValue to total
22    total = total + gradeValue;
23    // add 1 to gradeCounter
24    gradeCounter = gradeCounter + 1;
25  } // end while
26  // Termination Phase
27  average = total / 10; // calculate the average
28  // display average of exam grades
29  document.writeln("<h1>Class average is " + average + "</h1>" );
30  </script>
31  </head>
32  <body>
33    <p>Click Refresh (or Reload) to run the script again<p>
34  </body>
35  </html>
```

- `<html>`
- `<head>`
- `<title>Class Average Program</title>`
- `<script type = "text/javascript">`
- `var total; // sum of grades`
- `var gradeCounter; // number of grades entered`
- `var grade; // grade typed by user (as a string)`
- `var gradeValue; // grade value (converted to integer)`
- `var average; // average of all grades`
- `// Initialization Phase`
- `total = 0; // clear total`
- `gradeCounter = 1;`
- `// prepare to loop`
- `// Processing Phase`
- `// loop 5 times`
- `while (gradeCounter <= 5){`
- `// prompt for input and read grade from user`
- `grade = window.prompt("Enter integer grade:", "0");`
- `// convert grade from a string to an integer`

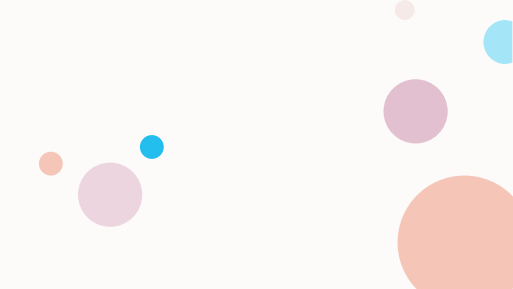
- gradeValue = parseInt(grade);
- // add gradeValue to total
- total = total + gradeValue;
- // add 1 to gradeCounter
- gradeCounter = gradeCounter + 1;
- } // end while
- // Termination Phase
- average = total / 5 ; // calculate the average
- // display average of exam grades
- document.writeln("<h1>Class average is " + average + "</h1>");
- </script>
- </head>
- <body>
- <p>Click Refresh (or Reload) to run the script again<p>
- </body>
- </html>

Code 13: Examination results calculation

```
1. <html xmlns>
2. <head>
3. <title>Analysis of Examination Results</title>
4. <script type = "text/javascript">
5. var passes = 0; // number of passes
6. var failures = 0; // number of failures
7. var student = 1; // student counter
8. var result; // one exam result
9. // process 5 students; counter-controlled loop
10. while ( student <= 5 )
11. {
12. result = window.prompt( "Enter result (1=pass,2=fail)", "0" );
13. if ( result == "1" )
14. passes = passes + 1;
15. else
```

```
16. failures = failures + 1;
17. student = student + 1;
18. } // end while
19. // termination phase
20. document.writeln( "<h1>Examination Results</h1>" );
21. document.writeln("Passed: " + passes + "<br />Failed: " + failures );
22. // -->
23. </script>
24. </head>
25. <body>
26. <p>Click Refresh (or Reload) to run the script again</p>
27. </body>
28. </html>
```



Assignment operators

- *Variable = variable operator expression;*
- This can be written in the form:
- `variable operator = expression;`

Arithmetic assignment operators

Assignment operator	Initial value of variable	Sample expression	Explanation	Assigns
<code>+=</code>	<code>c = 3</code>	<code>c += 7</code>	<code>c = c + 7</code>	10 to c
<code>-=</code>	<code>d = 5</code>	<code>d -= 4</code>	<code>d = d - 4</code>	1 to d
<code>*=</code>	<code>e = 4</code>	<code>e *= 5</code>	<code>e = e * 5</code>	20 to e
<code>/=</code>	<code>f = 6</code>	<code>f /= 3</code>	<code>f = f / 3</code>	2 to f
<code>%=</code>	<code>g = 12</code>	<code>g %= 9</code>	<code>g = g % 9</code>	3 to g

Increment and decrement operators

Operator	Example	Called	Explanation
++	++a	preincrement	Increment a by 1, then use the new value of a in the expression in which a resides.
++	a++	postincrement	Use the current value of a in the expression in which a resides, then increment a by 1.
--	--b	predecrement	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	b--	postdecrement	Use the current value of b in the expression in which b resides, then decrement b by 1.

1. <html>
2. <head>
3. <title>Preincrementing and Postincrementing</title>
4. <script type = "text/javascript">
5. var c;
6. c = 5;
7. document.writeln("<h3>Postincrementing</h3>");
8. document.writeln(c); // prints 5
9. // prints 5 then increments
10. document.writeln("
" + c++);
11. document.writeln("
" + c); // prints 6
12. c = 5;
13. document.writeln("<h3>Preincrementing</h3>");
14. document.writeln(c); // prints 5
15. // increments then prints 6
16. document.writeln("
" + ++c);
17. document.writeln("
" + c); // prints 6
18. </script>
19. </head><body></body>
20. </html>

Precedence and associativity of the operators

Operator	Associativity	Type
++ --	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

Counter controlled repetition

Code 15

```
<html>
<head>
  <title>Counter-Controlled Repetition</title>
  <script type = "text/javascript">

var counter = 1; // initialization
while ( counter <= 7 )
{
  document.writeln( "<p style = \"font-size: " +
counter + "em\">XHTML font size " + counter + "em</p>" );
  ++counter;    // increment
} //end while
</script>
</head><body></body>
</html>
```

Code 15

XHTML font size 1ex

XHTML font size 2ex

XHTML font size 3ex

XHTML font size 4ex

XHTML font size 5ex

XHTML font size 6ex

XHTML font size 7ex

for repetition statement : code 16

```
1. <html>
2. <head>
3. <title>Counter-Controlled Repetition</title>
4. <script type = "text/javascript">
5.   for ( var counter = 1; counter <= 7; ++counter )
6.     document.writeln( "<p style = \"font-size: \" + counter + \"ex\\\">XHTML
font size \" + counter + \"ex</p>" );
8. </script>
9. </head><body></body>
10. </html>
```

for repetition statement

- The for statement first line (including the keyword for and everything in parentheses after for) is often called the **for statement header**.
- If you incorrectly write `counter < 7`, the loop will execute only six times. This is an example of the common logic error called an **off-by-one error**.
- The general format of the **for** statement is
- **for** (*initialization*; *loopContinuationTest*; *increment*)
 statements

. for repetition statement

The diagram illustrates the syntax of a `for` loop with the example code: `for (var counter = 1; counter <= 7; ++counter)`. Annotations with arrows point to specific parts of the code:

- `for` is labeled as the *keyword*.
- `var counter` is labeled as the *Control variable name*.
- `= 1` is labeled as the *Initial value of control variable*.
- `counter <= 7` is grouped by a bracket and labeled as the *Loop-continuation condition*.
- `7` is labeled as the *Final value of control variable for which the condition is true*.
- `++counter` is labeled as the *Increment of control variable*.

```
for ( var counter = 1; counter <= 7; ++counter )
```

Code 17: sum the even integers from 2 to 100.

```
1. <html>
2. <head>
3. <title>Sum the Even Integers from 2 to 100</title>
4. <script type = "text/javascript">
5. var sum = 0;
6. for ( var number = 2; number <= 100; number += 2 )
7. sum += number;
8. document.writeln( "The sum of the even integers from 2 to 100 is " + sum );
9. </script>
10. </head><body></body>
11. </html>
```

Code 18: Compound interest calculation with a for loop

```
1. <html>
2. <head>
3. <title>Calculating Compound Interest</title>
4. <style type = "text/css">
5. table { width: 59% }
6. th { text-align: left }
7. </style>
8. <script type = "text/javascript">
9. var amount; // current amount of money
10. var principal = 1000.0; // principal amount
11. var rate = .05; // interest rate
12. document.writeln("<table border = \"5\">" ); // begin the table
13. document.writeln("<caption>Calculating Compound Interest</caption>" );
14. document.writeln("<thead><tr><th>Year</th>" ); // year column heading
```

```
15. document.writeln("<th>Amount on deposit</th>" ); // amount column heading
16. document.writeln( "</tr></thead><tbody>" );
17. // output a table row for each year
18. for ( var year = 1; year <= 10; ++year )
19. {
20. amount = principal * Math.pow( 1.0 + rate, year );
21. document.writeln( "<tr><td>" + year + "</td><td>" + amount.toFixed(2) + "</td></tr>" );
22. } //end for
23. document.writeln( "</tbody></table>" );
24. </script>
25. </head><body></body>
26. </html>
```

Code 18

Calculating Compound Interest

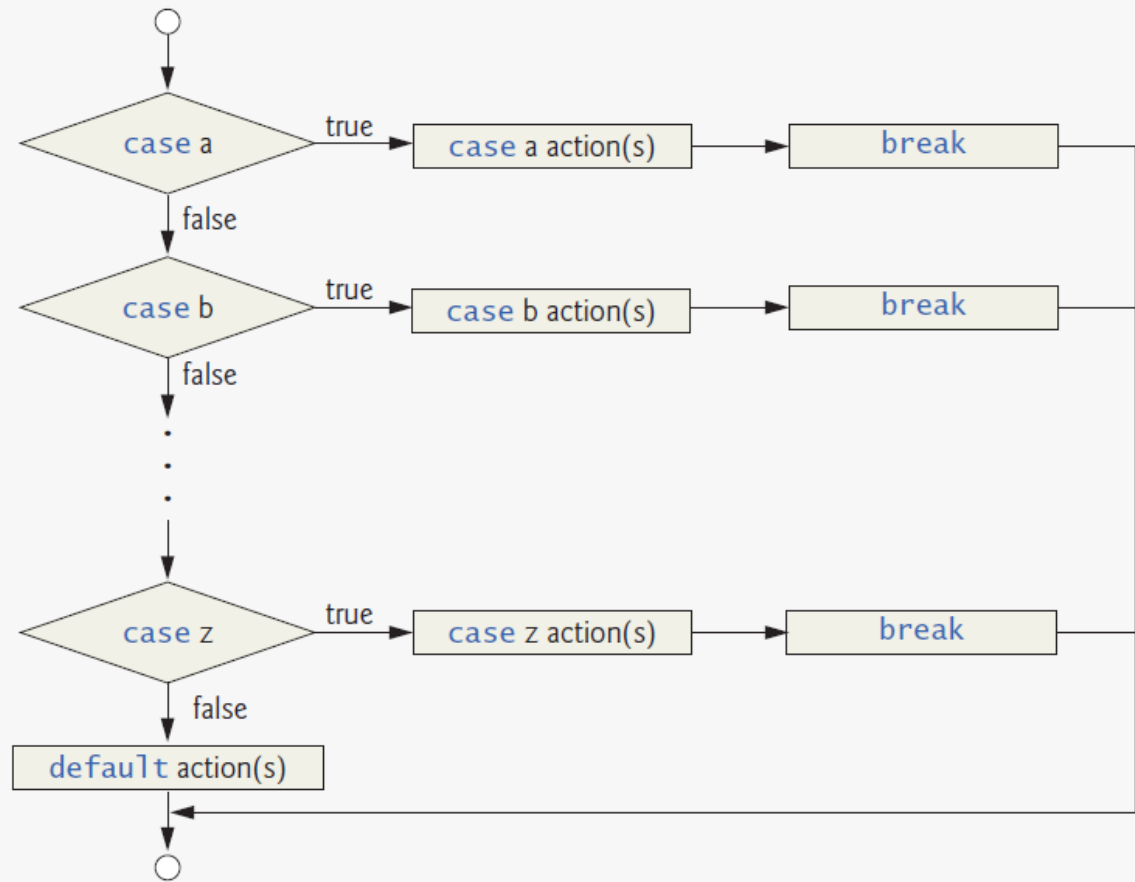
Year	Amount on deposit
1	1050.00
2	1102.50
3	1157.63
4	1215.51
5	1276.28
6	1340.10
7	1407.10
8	1477.46
9	1551.33
10	1628.89

Code 18: Compound interest calculation with a for loop


- `Math.pow(x, y)` calculates the value of `x` raised to the `y`th power.
- The `toFixed` method of a `Number` object formats the value by rounding it to the specified number of decimal places.

switch multiple selection statement

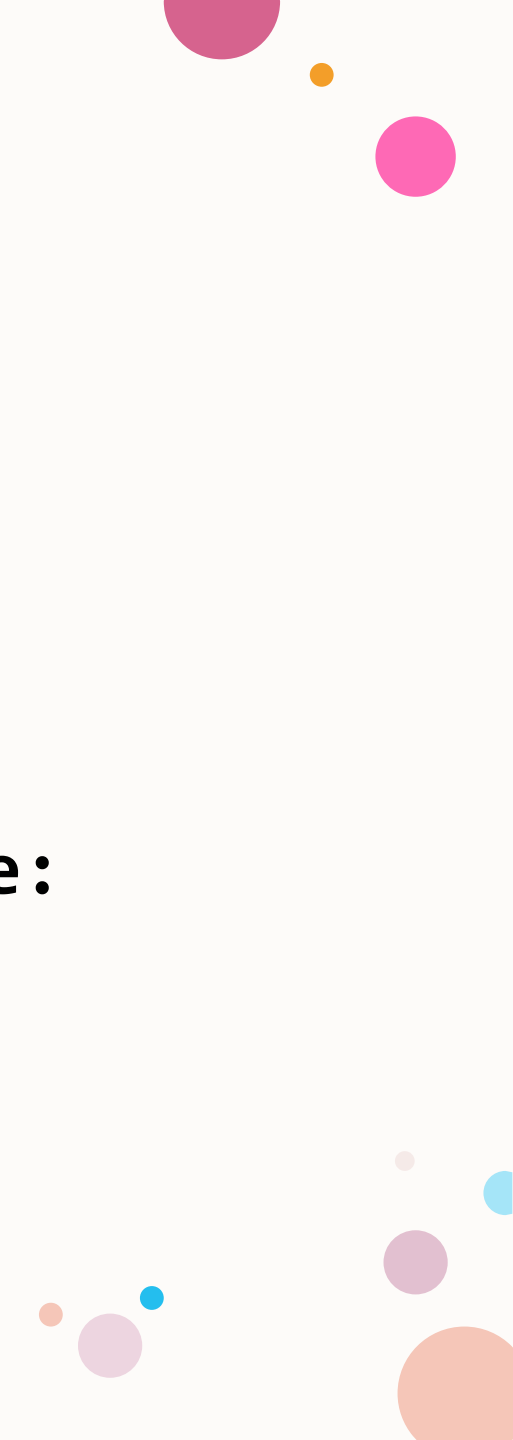
```
switch(expression) {  
  case x:  
    // code block  
    break;  
  case y:  
    // code block  
    break;  
  default:  
    // code block  
}
```



```
1 <html>
2 <head>
3 <title>Switching between XHTML List Formats</title>
4 <script type = "text/javascript">
5   var choice; // user's choice
6   var startTag; // starting list item tag
7   var endTag; // ending list item tag
8   var validInput = true;
9   var listType; // type of list as a string
10  choice = window.prompt( "Select a list style:\n" +
    "1 (numbered), 2 (lettered), 3 (roman)", "1" );
```



```
11 switch ( choice )
12 {
13     case "1":
14         startTag = "<ol>";
15         endTag = "</ol>";
16         listType = "<h1>Numbered List</h1>";
17         break;
18     case "2":
19         startTag = "<ol style = \"list-style-type:
upper-alpha\">";
20         endTag = "</ol>"
21         listType = "<h1>Lettered List</h1>";
22         break;
```



```
23 case "3":
24     startTag = "<ol style = \"list-style-type:
upper-roman\">";
25     endTag = "</ol>"
26     listType = "<h1>Roman Numbered List</h1>";
27     break;
28 default:
29     validInput = false;
30 } //end switch
31 if ( validInput == true )
32 {document.writeln( listType + startTag );
```

```
33 for ( var i = 1; i <= 3; ++i )
34 document.writeln( "<li>List item " + i + "</li>" );
35 document.writeln( endTag );
36 } //end if
37 else
38 document.writeln( "Invalid choice: " + choice );
39 </script>
40 </head>
41 <body>
42 <p>Click Refresh (or Reload) to run the script
again</p>
43 </body>
44 </html>
```

Code 19

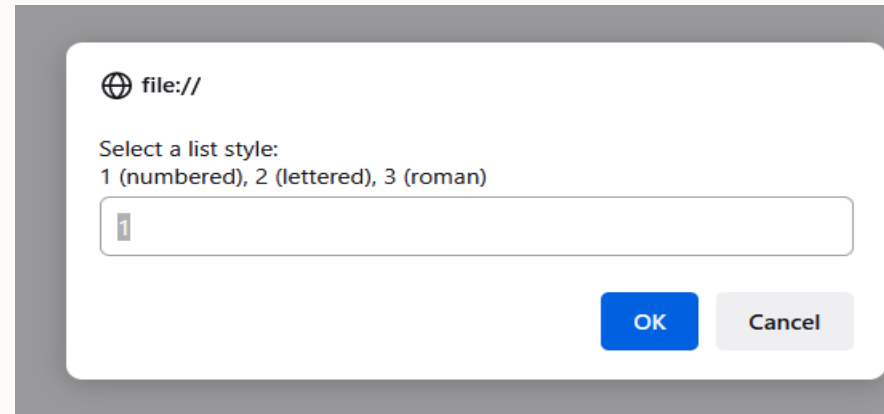
Roman Numbered List

- I. List item 1
- II. List item 2
- III. List item 3

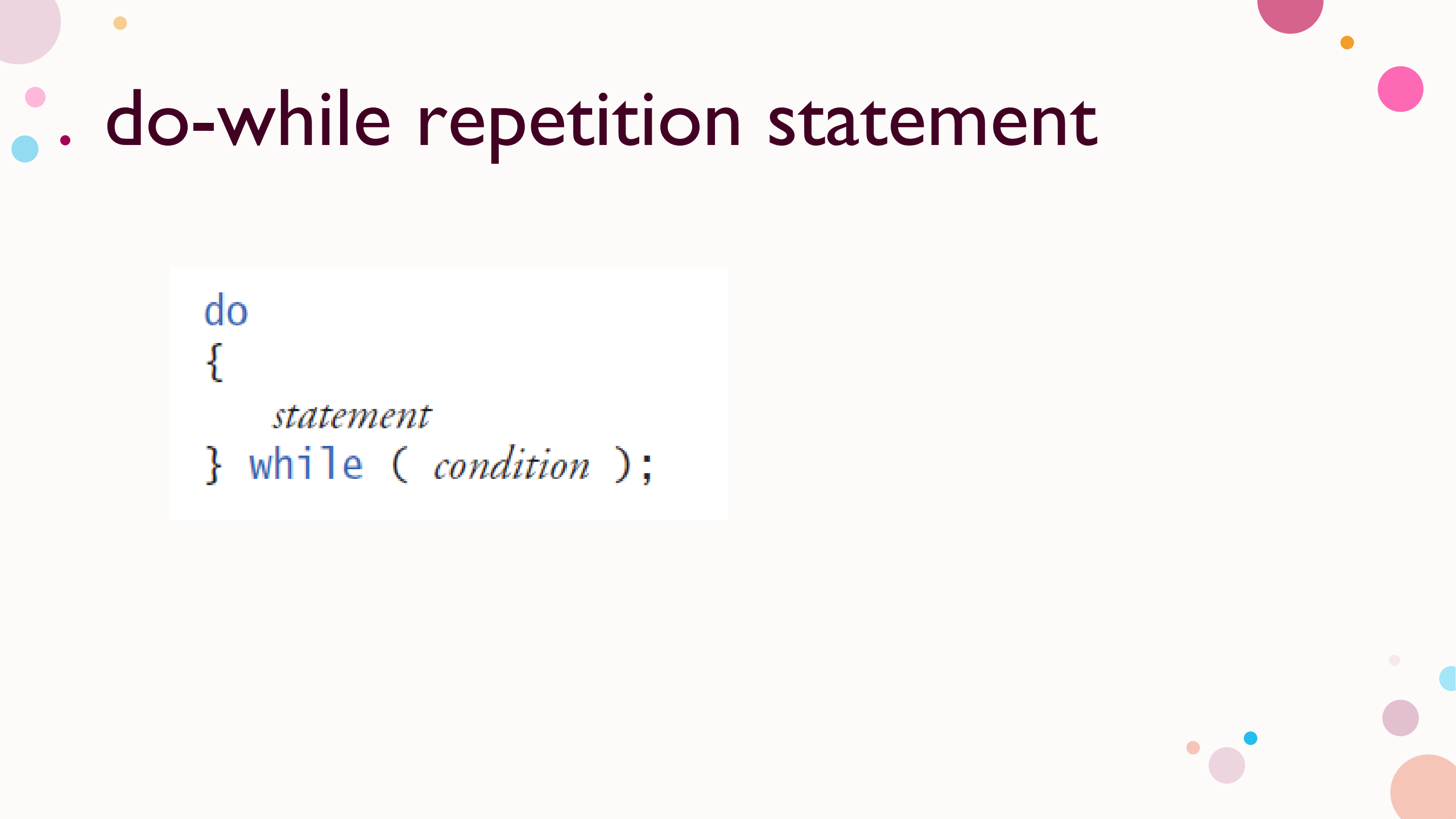
Click Refresh (or Reload) to run the script again

Invalid choice: 13

Click Refresh (or Reload) to run the script again



A screenshot of a web browser dialog box. The title bar shows a globe icon and the text "file://". The main content area contains the text "Select a list style:" followed by "1 (numbered), 2 (lettered), 3 (roman)". Below this is a text input field containing the number "1". At the bottom right of the dialog are two buttons: "OK" (blue) and "Cancel" (gray).



do-while repetition statement

```
do  
{  
    statement  
} while ( condition );
```

Code 20: do-while

- `<html>`
- `<head>`
- `<title>Using the do...while Repetition Statement</title>`
- `<script type = "text/javascript">`
- `var counter = 1;`
- `do {`
- `document.writeln("<h" + counter + ">This is " +`
- `"an h" + counter + " level head" + "</h" +`
- `counter + ">");`
- `++counter;`
- `} while (counter <= 6);`
- `</script>`
- `</head><body></body>`
- `</html>`



Code 20

This is an h1 level head

This is an h2 level head

This is an h3 level head

This is an h4 level head

This is an h5 level head

This is an h6 level head



• Break and continue statement

- The break statement "jumps out" of a loop.
- The continue statement "jumps over" one iteration in the loop.
- The **continue** statement, when executed in a **while**, **for** or **do...while** statement, skips the remaining statements in the body of the statement and proceeds with the next iteration of the loop.

1. <html>
2. <head>
3. <script type = "text/javascript">
4. for (var count = 1; count <= 10; ++count)
5. {
6. if (count == 5)
7. break;
8. document.writeln("Count is: " + count + "
");
9. }
10. document.writeln("Broke out of loop at count = " + count);
11. </script>
12. </head><body></body>
13. </html>

Code 21

```
Count is: 1  
Count is: 2  
Count is: 3  
Count is: 4  
Broke out of loop at count = 5
```

1. <html>
2. <head>
3. <script type = "text/javascript">
4. for (var count = 1; count <= 10; ++count)
5. {
6. if (count == 5)
7. continue; // skip remaining loop code only if count == 5
8. document.writeln("Count is: " + count + "
");
9. }
10. document.writeln("Used continue to skip printing 5");
11. </script>
12. </head><body></body>
13. </html>



Code 22

```
Count is: 1  
Count is: 2  
Count is: 3  
Count is: 4  
Count is: 6  
Count is: 7  
Count is: 8  
Count is: 9  
Count is: 10  
Used continue to skip printing 5
```

Labeled break and continue

```
<html>
<body>
<h2>JavaScript break</h2>
<p id="demo"></p>
<script>
const cars = ["BMW", "Volvo", "Saab", "Ford"];
let text = "";
list: {
  text += cars[0] + "<br>";
  text += cars[1] + "<br>";
  break list;
  text += cars[2] + "<br>";
  text += cars[3] + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script> </body> </html>
```



output

JavaScript break

BMW
Volvo

Code 23: write js to get a pattern shown below using labeled break statement

```
*****  
*****  
*****  
*****
```

End of script

Code 23: write js to get a pattern shown below using labeled continue statement



Logical operators : code 24

```
<html>
```

```
<body>
```

```
<h2>The && Operator (Logical AND)</h2>
```

```
<p>The && operator returns true if both expressions are true, otherwise it returns false.</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
let x = 6;
```

```
let y = 3;
```

```
document.getElementById("demo").innerHTML =
```

```
(x < 10 && y > 1) + "<br>" +
```

```
(x < 10 && y < 1);
```

```
</script>
```

```
</body>
```

```
</html>
```

Code 24

The && Operator (Logical AND)

The && operator returns true if both expressions are true, otherwise it returns false.

```
true  
false
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>Use of The class Attribute in JavaScript</h2>
```

```
<p>Click the button to change the color of all  
elements with class name "city":</p>
```

```
<button onclick="myFunction()">change  
colour</button>
```

```
<h2 class="city">London</h2>
```

```
<p>London is the capital of England.</p>
```

```
<h2 class="city">Paris</h2>
```

```
<p>Paris is the capital of France.</p>
```

```
<h2 class="city">Tokyo</h2>
```

```
<p>Tokyo is the capital of Japan.</p>
```

```
<script>
```

```
function myFunction() {
```

```
    var x =  
    document.getElementsByClassName("city");
```

```
    for (var i = 0; i < x.length; i++) {
```

```
        x[i].style.color = "blue";
```

```
    }
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```