SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10005 Web Development

Module 8 – JavaScript Part 2

Contents

- JavaScript Basic Syntax
- Functions
  - Function definition, parameters, call and return
  - Variable scope
- Statements
  - Selection
  - Loop

JAVASCRIPT BASIC SYNTAX

JavaScript Basic Syntax

- Is case sensitive

```
var firstName;
firstname = "Michael";
```

- Uses semicolons to separate statements

```
function greet() {
    alert("This is Nathan.");
    alert("How are you?");
    alert("How can I help you?");
}
```

- Uses comma to separate words in a list

```
var firstName, lastName;
```

JavaScript Basic Syntax

- Uses round brackets ( ) for operation precedence and argument lists

```
result=(3+5)*((4-2)/2)
```

- Uses curly or brace brackets { } for blocks of code

```
function greet() {
    alert("This is Nathan.");
    alert("How are you?");
    alert("How can I help  you?");
}
```

JavaScript Basic Syntax

- Has *Keywords* (reserved words) that have special meanings within the language syntax, such as

  abstract boolean break byte case catch char class const continue debugger default delete do double else enum export extends false final finally float for function goto if implements import in instanceof int interface long native new null package private protected public return short static super switch synchronized this throw throws transient true try typeof undefined var void volatile while with

JavaScript

- In-built JavaScript functions
- **`alert`** - Displays an output through a window

```
alert("Your password is wrong!");
```

- **`prompt`** - Displays a prompt and returns an input from the user through the keyboard

```
var name =
  prompt("Please enter your name:");
```

FUNCTIONS

Defining Functions

- **Functions** are groups of statements that you can execute as a single unit
- **Function definitions** are the lines of code that make up a function
- The syntax for defining a function is:

```
function nameOfFunction(parameters) {
        statements;
}
```

For example:

```
function GST(amount) {
        var result = amount/11;
        alert(result);
}
```

Defining Functions (continued)

- A **parameter** is a variable used within a function
- Parameters are placed within the parentheses that follow the function name

```
function GST(amount) {
    var result = amount/11;
    alert(result);
}
```

- Functions **do not** have to contain parameters
- Functions enable reusability of code

```
GST(220);
GST(40+300);
gst(100);
```

Defining Functions (continued)

- The pair of curly braces (called **function braces**) contain the function statements
- **Function statements** do the actual work of the function and must be contained within the function braces

```
function showName(fName, lName) {
    var result = fName + lName;
    alert(result);
}
```

concatenates two strings

Calling Functions

- Function must be **called** using the **function name** with **()** in order to be executed

```
alert("Web development");
showName("Web", "Development");
```

Calling Functions

- `()` is required even when a function is called if the function has no parameters

```
function printWelcome() {
   alert ("Welcome!");
}
printWelcome();
```

Returning Values

- A **return** statement in a function is a statement that returns a value to the statement that called the function
- A function does not necessarily have to return a value

```
function averageNumbers(a, b, c) {
        var sum = a + b + c;
        var result = sum / 3;
        return result;
}
```

Returning Values (continued)

- Functions that return values, work like an expression, usually with an assignment operator.
- For example

```
var x = averageNumbers(3, 4, 5);
```

To display the result, the alert function can be used

```
var x = averageNumbers(3, 4, 5);
alert(x);
```

OR

```
alert(averageNumbers(3, 4, 5));
```

Understanding Variable Scope

- **Variable scope** is “where in your program” a declared variable can be used
- A variable’s scope can be either **global** or **local**
- A **global variable** is one that is declared outside a function and is available to all parts of your program
- A **local variable** is declared using the **var** keyword inside a function and is only available within the function in which it is declared

Understanding Variable Scope (Cont.)

```
// script.js
var globalVariable = "Global";

function testScope1() {
  var localVariable = "Local";
  alert(localVariable);
  alert(globalVariable);
}

function testScope2() {
  alert(localVariable);
  alert(globalVariable);
}
```

Understanding Variable Scope (Cont.)

```
// variables must be declared before they can be
   used
//script.js
globalVariable = "Global";

function testScope1() {
  var localVariable="Local";   //variable defined

  alert(localVariable);        //variable used
}
```

Output

Understanding Variable Scope (Cont.)

```
// variables must be declared before they can be
   used
//script.js
function testScope1() {
  alert(localVariable);
  alert(globalVariable);
}
```

Error, as *localVariable* and *globalVariable* are not defined.

No Output!

JAVASCRIPT STATEMENTS

Comments in Javascript

- Single Line

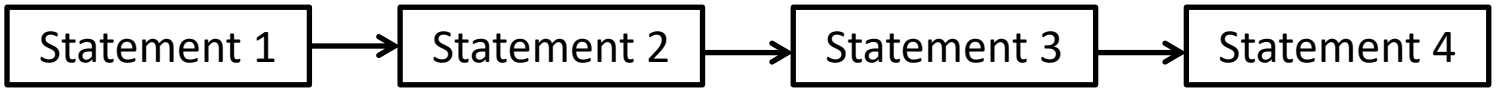
```
// begins a single line comment
```

- Multi-line comment:

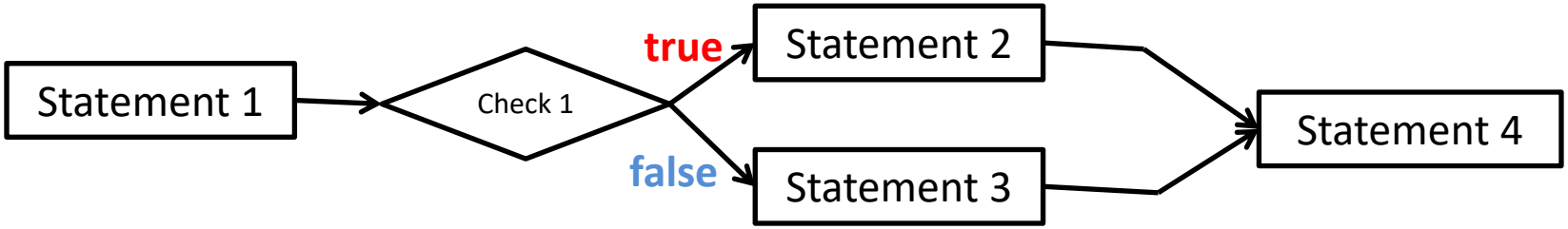
```
/*
   this is a
   multi-line comment
*/
```

Three Models in Programming

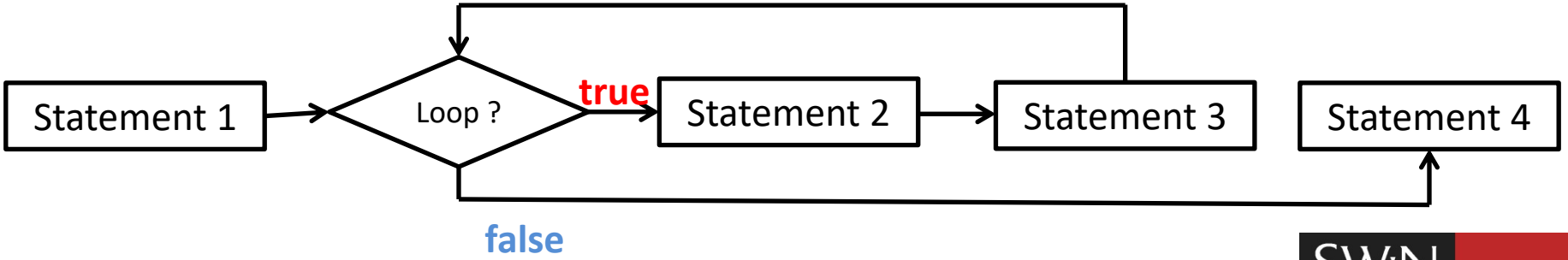
- Sequence



- Selection



- Repetition



Sequence

- Statements in a function are executed in sequence, one line at a time.
- When functions are called, execution jumps into the function. Once the function completes its execution, the program continues to execute the next line of code after the call.
- Functions are not executed, unless called.

Sequence (continued)

START

```
1  function function1() {
4    var name="Derpina";
5    function2(name);
6    alert("That is sad.");
7  }
8
9  function function2(n) {
10   alert("My name is "+n);
11 }
```

Selection

- **Selection** is the process of determining which statement to execute in a program.
- There are two types of selection
  - `if`
  - `switch`

Selection – `if` statement

- Use the `if` statement to execute a block of statements if a logical *condition* is true.
- Use the optional `else` clause to execute a further statement if the *condition* is false.

true

false

```
if (a>b) {
  alert("a is bigger!"); //statement_1
}else {
  alert("b is bigger!"); //statement_2
}
alert("THE END");
```

The else clause is optional.

Selection – `if` statement (continued)

- A `condition` can be any expression that evaluates to **true** or **false**.
  - `if(a>b)` `if(a==b)` `if(a>=6)`
- If `condition` evaluates to **true**, `statement_1` is executed. Otherwise `statement_2` is executed.
  Only one of the two statements is executed. Execution is no longer **sequence**.
- `statement_1` and `statement_2` can be *any* statement, including further **nested** `if` statements.

Selection – `if` statement (continued)

- **else if**

```
if (condition_1) {
    statement_1;
} else if (condition_2){
    statement_2;
} else if (condition_3) {
    stetement_3;
} else {
  statement_n;
}
```

You can have a number of **else if** statements

Selection – `if` statement (continued)

```
var score;
score = prompt("Enter a score:");
if (score == 10){
  alert("Score is 10");
} else if (score > 10){
  alert("Score is greater than 10");
} else {
  alert("Score is less than 10");
}
```

Selection – `if` statement (continued)

```
var n = prompt("Enter a score:");
//var ans = "";

if (n >= 80 && n <= 100){
  result = "HD";
} else if (n >= 70 && n < 80) {
  result = "D";
} else if (n >= 60 && n < 69) {
  result = "C";
} else if (n >= 50 && n < 59) {
  result = "P";
} else if (n >= 0 && n < 50) {
  result = "F";
} else {
  result = "Null or invalid score.");
}
alert("You obtained a " + result);
```

Selection – `switch` statement

- A **`switch`** statement allows a program to evaluate an expression and attempt to match the expression's value to a **`case`**.
- If a match is found, the program executes the associated block of statements.

Selection – `switch` statement (cont)

```
switch (expression) {
    case case_label_1:
        statements_1;
        [break;]
    case case_label_2:
        statements_2;
        [break;]
    ...
    [default:
        statements_def;
        alert("Invalid data!");
    ]
}
```

Optional

Optional

Selection – `switch` statement (cont)

- The program first looks for a **case** clause with a label matching the value of expression and then transfers control to that clause, executing the associated statements.
- If no matching label is found, the program looks for the optional **default** clause, and if found, transfers control to that clause, executing the associated statements.
- If no default clause is found, the program continue to execute the statement following the end of **switch**.

Selection – `switch` statement (cont)

- By convention, the **`default`** clause is the last clause.
- The optional **`break`** statement associated with each case clause ensures that the program *breaks* out of **`switch`** once the matched statement is executed and continues to execute the statement *following* the **`switch`**.
- If **`break`** is omitted, the program continues to execute the next statement *in the* ***`switch`*** statement.

Selection – `switch` statement (cont)

```
var result = "";
var fruitType = prompt("Enter a fruit");

switch (fruitType) {
case "Oranges":
      result = "Oranges are $3.00 a kilo.";
      break;
case "Apples":
      result = "Apples are $1.99 a kilo.";
      break;
case "Mangoes":
case "Bananas":
      result = "Mangos and bananas are $2.00 each.";
      break;
default:
      result = "Sorry, we are out of " + fruitType;
}
alert(result);
```

Selection – `switch` statement (cont)

- In the example, if `fruitType` evaluates to "Oranges", the program matches the value with case "Oranges" and executes the associated statement.
- When break is encountered, the program terminates **switch** and executes the statement following **switch**.
- If breaks were omitted, the statement for case "Apples" would also be executed.

Repetition

- Repetition is expressed using **loop** statements
- A **loop** statement is a control structure that repeatedly executes a statement or a block of statements while a specific condition is ***true*** or until a specific condition becomes ***true***
- There are four types of **loop** statements:
  - **`while`** statements
  - **`do while`** statements
  - **`for`** statements
  - **`for in`** statements

Repetition – `while` loop

- A **while** loop uses a ***pre-loop*** *condition test*. This means there is a possibility that the statements may never be executed.
- The condition test occurs *before* statements in the loop are executed. If the condition returns **true**, statement is executed and the condition is tested again. If the condition returns **false**, execution stops and the program continues to execute the statement following the while loop.

Repetition – `while` statement (cont)

START

true

false

```
while (condition){
    statements;
}
statement_01;
statement_02;
```


SWINBURNE UNIVERSITY OF TECHNOLOGY

Repetition – **while** statement (cont)

- Example

```
var i = 0;
var sum = 0;
while (i < 3){
    sum = sum + i;
    i = i + 1;
}
alert(sum);
```

With each iteration, the loop increments i and adds that value to sum.
Therefore, i and sum take on the following values:
After the first iteration: i = 1 sum = 0
After the second iteration: i = 2 sum = 1
After the third iteration: i = 3 sum = 3
After completing the third iteration, the condition i < 3 is no longer true, so the loop terminates.

- *What will be displayed?*

Repetition – `do-while` statement

- A **`do while`** loop uses a ***post-loop*** *condition test*, which means the statements in the loop block will be executed at least once.
- If `condition` is true, the code block in the loop will be executed again. At the end of every execution, the condition is checked.

Repetition – **do while** statement (cont)

- When the `condition` is **`false`**, the loop stops and the program continues to execute the statement following the `do while` loop.

START

```
do {
    statements;
} while (condition);
statement_01;
statement_02;
```

true

false

SWINBURNE UNIVERSITY OF TECHNOLOGY

Repetition – **do while** statement (cont)

- Example

```
var i = 0;
var sum = 0;
do {
   sum = sum + i;
   i = i + 1;
} while (i < 3);
alert(sum);
```

What will be displayed?

Repetition – **for** statement

- A **for** loop repeats until the condition evaluates to **false**.
- A **for** loop can be repeated for 0, 1 or many times.

```
for ([initialisation];
        [condition];[update]) {
   statements;
}
```

For example:

```
for (var i=0; i<10; ++i) {…}
```

Repetition – `for` statement (continued)

- When a **for** loop executes, the following occurs:
- The `initialisation` expression if any, is executed.
  - This expression usually initialises a loop counter.

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

  - A variable can be declared and initialised in this expression.
  - `for (var i=0; i<10; ++i) {…}`

Repetition – `for` statement (continued)

- The `condition` represents a check that determines if the loop is repeated.
  - If the `condition` is **true**, the code block in the loop is executed.
  - If the `condition` is false, the `for` loop terminates.

```
for (var i=0; i<10; ++i) {…}
```

- The `update` expression, if there is one, executes, and control returns to `condition`

Repetition – `for` statement (continued)

- The loop repeats as long as the `condition` remains true.

START

```
for ([initialisation];
     [condition];[update])
{
    statements;
}
statement_1;
Statement_2;
```

false

true

SWINBURNE UNIVERSITY OF TECHNOLOGY

Repetition – `for` statement (continued)

- Example

```
var i;
var sum = 0;
for (i = 1; i < 3; i++) {
    sum = sum + i;
}
alert(sum);
```

- *What will be displayed?*

Repetition – **`for in`** statement

- The **`for in`** loop is a special loop that allows easy traverse through a collection of elements.
- JavaScript provides a number of inbuilt collections which can be traversed using a **`for-in`** loop.

```
for (variable in collectionOfObject) {
    statements;
}
```

Repetition – `for-in` statement (cont)

- Example

```
var unit = "Internet Technologies";
var allUnits = ["Web Development",
                "Web Programming"];
var oneUnit;
var ans = "";
for (oneUnit in allUnits) {
  ans = ans + allUnits[oneUnit]);
}
alert(ans);
```

Collections will be discussed in the next lecture.

- *What will be displayed?*

NEXT LECTURE:

DOCUMENT OBJECT MODEL