

CS2204 Fundamentals of Internet Applications Development

6. Javascript Part I

6.1. Programming overview

- 6.1.1. Input
- 6.1.2. Output
- 6.1.3. Control Flow

6.2. Javascript Overview

- 6.2.1. How Javascript is used in this course?
- 6.2.2. How Javascript is taught in this course?
- 6.2.3. Characteristics
- 6.2.4. Execution Environment
- 6.2.5. How to run Javascript?
 - 6.2.5.1. Embedded Script
 - 6.2.5.2. External Script
 - 6.2.5.3. Inline Script

6.3. Javascript Basic

- 6.3.1. Statement
- 6.3.2. Comments
- 6.3.3. Execution Statement
- 6.3.4. Variable
 - 6.3.4.1. Variable and type in Javascript
 - 6.3.4.2. Primitive Types
 - 6.3.4.3. Variable Declaration
 - 6.3.4.4. Variable Scope
 - 6.3.4.5. Common Traps about Variable Scope
 - 6.3.4.6. Let
 - 6.3.4.7. Const
- 6.3.5. Expression
 - 6.3.5.1. Arithmetic Operators
 - 6.3.5.2. Special behaviour of String
 - 6.3.5.3. Assignment Operators
 - 6.3.5.4. Comparison & Logical Operators

- 6.3.5.5. Operator Reference
- 6.3.6. Flow Control Statements
 - 6.3.6.1. If-else
 - 6.3.6.2. Switch
 - 6.3.6.3. For loop
 - 6.3.6.4. While & do-while loop
 - 6.3.6.5. Break, continue, return & block
- 6.3.7. Debugging
- 6.3.8. Javascript Functions
 - 6.3.8.1. Function Declaration
 - 6.3.8.2. Function Parameter
 - 6.3.8.3. Special Characteristics of Function
 - 6.3.8.4. Built-in Javascript Function
- 6.4. Conclusions**

6. Javascript Part I

3 main topics in this part.

Programming and Javascript Overview

- a very brief introduction to programming as some of you may not have learnt programming before
- introduce how Javascript is used in this course
- highlight some difference between Javascript and other common programming languages

How to run Javascript in Web pages

- embedded
- external
- inline
- event driven (**Reactive Programming**)

6.1. Programming overview

6.1. Programming overview

Program is a set of instructions to tell a computer what to do which usually processes data :

- from user input or storage (hard disk or memory)
- process the data, may use temporary locations in memory (variable) to store result
- output to screen, storage or in our case Web page
- Javascript is a bit different, it **tells the browser what to do**

Each instruction is a statement for :

- defining or assigning values to variable
- doing something (e.g. call a function or method – to be learnt later)
- making decision
- repeating some operations

Writing a program is then to :

- think of a way to solve the problem first (i.e. the logic or algorithm)
- build up instructions with **flow control** according to the designed algorithm

6.1.1. Input

6.1.1. Input

Simplest way is to get data from keyboard, different ways to do that

```
var r = confirm("Press a button");  
if (r == true) {x = "You pressed OK!";}   
else {x = "You pressed Cancel!";} 
```

```
var person = prompt("Please enter your name", "Harry Potter");  
if (person != null) {alert("Hello, " + person);} 
```

Fill in forms is also a way to input data. clicking, right-click or scrolling can all be regarded as input.

6.1.2. Output | **Confirm** - http://www.w3schools.com/js/tryit.asp?filename=tryjs_confirm | **Prompt** - http://www.w3schools.com/js/tryit.asp?filename=tryjs_prompt

6.1.2. Output

Simplest way is to show up pop-up message box

```
alert("hello world");
```

```
var name="M T Chan";  
alert("hello " + name);
```

Write text/html into the Web page

```
document.write("hello world");  
document.write("<h1>writing a heading with Javascript</h1>");
```

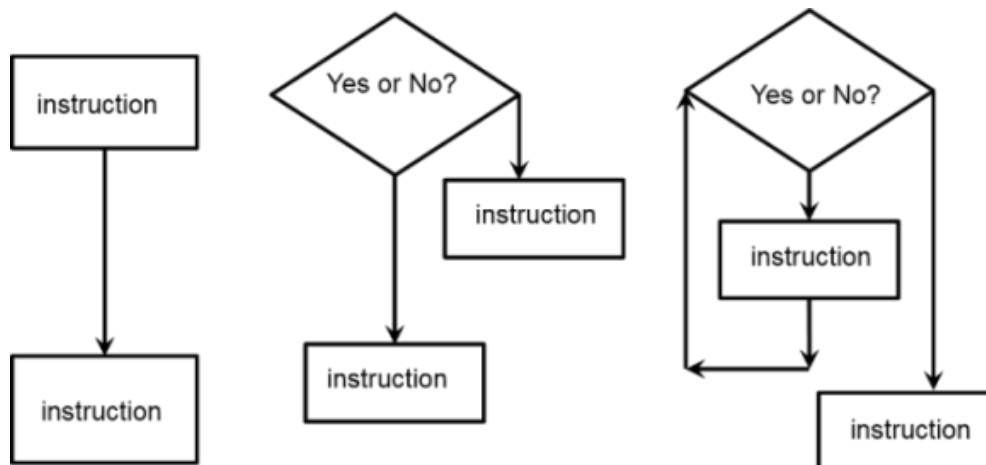
More ways to show output later ...

6.1.3. Control Flow

Basic control flow can be:

- sequential (i.e. one instruction after another)
- set up different paths based on conditions (if-then-else)
- repeating instructions for a number of times or depending on some conditions (loop)

these in principle can represent all algorithms but some more types of flow can make programming easier.



6.2. Javascript Overview

6.2. Javascript Overview

Historical development

- JavaScript is created by Netscape Communications Corporation, not related to Java
- Microsoft has its own implementation of the language as Jscript, compatible with JavaScript
- the Chrome V8 Javascript engine is now very popular because of Node.js (the server side script environment)
- has been standardized by the European Computer Manufacturers Association (ECMA), unlike HTML and CSS, JavaScript is not a standard maintained by the W3C

JS under the management of ECMA has had only a few specification revision:

- late 1996 after first release, submitted to ECMA as standard 1st edition
- 1999 significant update to 3rd edition
- 2009 5th edition, sometimes known as ECMAScript 5 (4th edition abandoned)

- 2015 latest 6th edition, ECMAScript 6 or ECMAScript 2015

JavaScript is object-based, originally designed to be used in Web pages which are written in HTML. It is therefore not intended to be self-sufficient but work under a host environment (e.g. the browser). It was designed as a Web Scripting Language for both the client and server sides. Client side scripts run in browser while server side scripts now mostly run in Node.js (built with V8 engine, similar to Java Virtual Machine (JVM) which has full access to the operating system environment.

6.2.1. How Javascript is used in this course? | ECMA - <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

6.2.1. How Javascript is used in this course?

Javascript

- is a set of instructions to tell a browser what to do
- with some characteristics different from other programming languages

Program Web pages

- do something HTML & CSS (especially) cannot do
- check user input, usually in forms
- respond to events

DHTML (dynamic HTML) & HTML5 features

- DHTML: combine the power of CSS and Javascript to enhance user experience; create HTML elements with JS
- HTML5 features: scripted audio & video; canvas

6.2.2. How Javascript is taught in this course? | demo 1 -

http://rainbow.arch.scriptmania.com/scripts/cursor_bubbles_trail.html

6.2.2. How Javascript is taught in this course?

Not as a full programming language

- just enough to work with Web pages
- as an introduction so that you can go further on your own later
- similar to CSS & HTML - we focus on the elements
- this is the 3rd S in the 3S: Structure, Style & Script - use JS to manipulate or create elements, respond to events occurred for elements

Select an object in the DOM

- use JS to manipulate it
- attach JS to it (**event handler/listener**)

■ identify events of an element

- execute a JS (event handlers) when they occur

6.2.3. Characteristics

6.2.3. Characteristics

JavaScript is now a full-featured object-oriented scripting language.

Considered lightweight in the past:

- object based in the past, now fully object-oriented covering inheritance and polymorphism
- very easy to start using it - just generate a Web page and run ... can be productive with it very quickly (but difficult to become a master)
- thought there is not a great deal to learn? (the most misunderstood language)

It is a scripting language as:

- tells an application (in our case, it is the browser) what to do
- cannot do much without the application (the script engine) providing the execution environment

6.2.4. Execution Environment

6.2.4. Execution Environment

Host environment for client scripting is Browser, providing:

- interface objects : windows, menus, pop-ups, anchors, frames, ... etc.
- a mean to attach script to events : change of focus, page/image load & unload, error, form submission, etc.

Host environment for server scripting is Script Engine (e.g. Node.js), providing:

- operating system API
- full networking features
- can use JS to implement a Web Server

Host environment of Web server (e.g. express.js) should provide:

- requests, clients and files

- mechanisms to share and lock data

ECMA Scripts therefore only defines **Native objects** (built-in, i.e. defined in the language) and leaves **Host objects** definition to the scripting engine.

6.2.5. How to run Javascript?

6.2.5. How to run Javascript?

Javascript is put inside Web page, the execution order is according to the order presented to the browser.

Can be :

- Embedded
- External
- Inline
- Event driven (3 ways to setup)

6.2.5.1. Embedded Script

Put in <head> section :

- can be any kind of statements
- scripts that contain functions usually go in the head section of the document
- it ensures that scripts are loaded before the function is called
- scripts can be used by elements in the entire web page

Inside <script> tag

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

6.2.5.2. External Script

Scripts are saved in an external JavaScript file, with .js file extension.

Advantages

- easier maintenance
- search engine friendly
- use codes from others (function libraries)
- provide codes for others (other pages)

```
<script type="text/javascript" src="... external JS file ..." >  
</script>
```

6.2.5.3. Inline Script

6.2.5.3. Inline Script

Scripts that are placed in the body section inside `<script> </script>`. Some scripts may exist inside attributes, e.g.

```
<h1 onclick="return clickme();">Click this heading</h1>
```

most people don't regard this as inline script.

Must use script tag or event attribute - cannot put inside an element as content

```
<h1>alert( )</h1>
```

this is wrong!

Advantages:

- useful for diagnostics (display alert messages)
- create HTML during page load depending on different situations, e.g. get current date time

Disadvantages

- difficult to update

6.3. Javascript Basic | **Example - <http://courses.cs.cityu.edu.hk/cs2204/example/html/21-JSCreateHTML.html>**

6.3. Javascript Basic

Javascript is composed of statements & comments. There are different types of statements. Remember that Javascript is **case sensitive**, i.e. MyVarName is not the same as myVarName.

Execution Statement

- assignment statement : variables, expressions, operators & operands
- function/method call

Flow Control Statement

Statements can also be grouped as a unit and defined as functions

- self-defined function
- built-in function (defined by Javascript and can be used without definition)

6.3.1. Statement

6.3.1. Statement

Each instruction is a JavaScript statement

- consists of keywords, e.g. if else; for, etc., keywords are reserved words, should not be used for function or variable names
- a single statement may span multiple lines
- multiple statements may occur on a single line if each statement is separated by a semicolon (;) – **not a good practice**

Usually all instructions as a whole form a program but Javascript in Web pages is a bit different

- usually not as one single program
- may spread out as different fragments
- each fragment enclosed by <script> ... </script> or in a separate file (depending on whether embedded, inline and/or external scripts are used)

6.3.2. Comments

6.3.2. Comments

A comment is any text, such as notes, clarification, or explanation of code. JavaScript will not interpret comments as script commands.

- Single line comment
- A line starts with //
- Example:
`// here is a single line comment`

Multi-line comment

- The /* at the beginning of the line tells JavaScript to ignore everything that follows until the end of the comment */
- Example:
`/* Use this method to
contain multiple lines of comments */`

6.3.3. Execution Statement

6.3.3. Execution Statement

Statements that actually do something (executable). Broadly divided into 3 kinds.

Creation/declaration of variable
`var myvariable;`

2 types of expressions

- formed with operator and operands, e.g. assign a value to variable

`x = y;`

(x,y are called operands and =: is the operator)

- function/method call - set of statements packaged as a unit for easy (re)use; functions defined in other objects (usually called **method** in Object-Oriented programming)

6.3.4. Variable

Variable

- in programming, used to store data value temporarily
- can create with a name, read or change its value

A few rules for variable names:

- must begin all variable names with either a letter or an underscore (_); some people use _ for global variable name prefix (explain more later) or **CamelCase** for other variables, e.g. ThisMyVariable
- can then use letter, digits, or underscore for all subsequent characters
- letters include all uppercase characters (A through Z), and lowercase characters
- digits include the characters 0 through 9
- there must not be any space in the variable name

No JavaScript reserved words are allowed, such as **break**, **export**, **this**. List of reserved words can be found at the link below.


6.3.4.1. Variable and type in Javascript | JS reserved words - https://www.w3schools.com/js/js_reserved.asp

6.3.4.1. Variable and type in Javascript

An object in Javascript belongs to a type which determines the object behaviour. A variable is the name of (or points to) an object and therefore the type applies to variable as well. There are 3 basic (**primitive**) types and other complex types:

- boolean
- number
- string
- other types : array, date, object, function, etc.

The type affects a variable/object's behavior & operation, e.g.

- string (usually) not used in arithmetic – more about this later
 - boolean only has two possible values – true or false
 - integer number does not have decimal point
- 

ther more complex types will be discussed later

- array (easier to understand as a list of boxes)
- date
- Math, etc.

6.3.4.2. Primitive Types

6.3.4.2. Primitive Types

Boolean – can contain two values **true** and **false** only

Number

- Integer – represented in JavaScript in decimal – 33, hexadecimal – 0x7b8, 0X395 or octal – 071
- Float – can be represented in either standard or scientific notation, e.g. 305.673, 8.32e+11, 1.2e2, 9.98E-12

String

- a sequence of zero or more characters.
- enclosed by double quote (") or single quote (')
- the quotes should be used with care; start double(single) quote matches with end double(single) quote; only single quote can be inside double quote and vice versa

6.3.4.3. Variable Declaration

6.3.4.3. Variable Declaration

Variable **needs to be** declared, in one of the following ways.

With var command

```
var alertmsg;  
var validity;
```

With value assignment

```
alertMsg = "The following error(s) is/are found.";  
validity = 365;
```

This is the most **misunderstood concept about declaration** in Javascript. It appears the variable alertMsg or validity does not require declaration and can be used directly. In fact Javascript declares/creates the variable for you if it is an assignment statement. If you use the variable in other statement without doing an assignment before, the variable is undefined, e.g.

```
alert(validity + 1);
```



Declare and assign in one statement
`var validity = 365;`

6.3.4.4. Variable Scope

6.3.4.4. Variable Scope

Variable Scope is the possible locations (in a Web page) where the variable can be referred to, used or valid. There are only two scopes, global and block. A block can be { }, a function (inside or outside) or even a statement. Variable declared in a block is **local** to that block. Local, outer block and global variables can be accessed.

```
var samename;  
function f() {  
    samename=samename + 1; //using the global variable outside the function  
}  
function g() {  
    var samename=1; //using the newly declared local variable,  
    //will be gone when function g finishes  
}
```

Variables cannot be used in HTML since HTML cannot read them, they are used in JavaScript only. On the other hand, Javascript can use HTML definition through the DOM (to be discussed later).

```
<script>var myvar=1; </script>  
<!-- this is wrong -->  
<h1 id=myvar>Try to get variable declared in JS</h1>
```

6.3.4.5. Common Traps about Variable Scope | variable scope -
<http://courses.cs.cityu.edu.hk/cs2204/example/html/27-JSVariableScope.html>

6.3.4.5. Common Traps about Variable Scope

Undefined because of unassigned

```
x = 1;  
alert(x);  
alert(y);
```

Variable within a function

```
var scope="global";  
function f() {  
    console.log(scope); //variable hoisting  
    var scope="local";  
    console.log(scope); //access local  
    localtoGlobal="ltg"; //implied global  
}  
f();
```

```
console.log(scope);  
console.log(localtoGlobal);
```

Result:

```
undefined  
local  
global  
ltoG
```

Scoping analysis:

- variable declared outside function is global
- declared inside function is local
- if inside function without using var, the variable is implied global
- **variable hoisting** means if any variable (especially local) is created by assignment or declaration, JS would implicitly make declaration at the top (**lexical scoping**)

6.3.4.6. let

Variable scoping is so confusing and demands discipline in programming, ECMAScript 6 introduces the concept of **block scope** and **let**. It declares variables that are limited in scope to the block, statement, or expression on which it is used.

```
function varTest() {  
    var x = 1;  
    if (true) {  
        var x = 2; // same variable!  
        console.log(x); // 2  
    }  
    console.log(x); // 2  
}  
  
function letTest() {  
    let x = 1;  
    if (true) {  
        let x = 2; // different variable  
        console.log(x); // 2  
    }  
}
```

```
    console.log(x); // 1  
}
```

6.3.4.7. const

6.3.4.7. const

Loose type in JS is another issue leading to error prone coding, `const` is introduced to declare a constant – a read-only reference to a value.

```
// NOTE: Constants can be declared with uppercase or lowercase  
//but a common convention is to use all-uppercase letters.
```

```
// define MY_FAV as a constant and give it the value 7  
const MY_FAV = 7;
```

```
// this will throw an error - Uncaught TypeError  
// Assignment to constant variable.  
MY_FAV = 20;
```

```
// MY_FAV is 7  
console.log('my favorite number is: ' + MY_FAV);
```

```
// trying to redeclare a constant throws an error  
const MY_FAV = 20;
```

```
// the name MY_FAV is reserved for constant above
// so this will fail too
var MY_FAV = 20;
```

```
// this throws an error too
let MY_FAV = 20;
```

 const can be either global or local according to the **block scope**

```
const MY_FAV=7;
// it's important to note the nature of block scoping
if (MY_FAV === 7) {
    // this is fine and creates a block scoped MY_FAV variable
    let MY_FAV = 20;

    // MY_FAV is now 20
    console.log('my favorite number is ' + MY_FAV);

    // this gets hoisted into the global context and throws an error
    var MY_FAV = 20;
}
```


Since `const` is a reference, although it cannot be re-assigned, its content can change if it is an object or array

```
// const also works on objects
```

```
const MY_OBJECT = {'key': 'value'};
```

```
// Attempting to overwrite the object throws an error
```

```
MY_OBJECT = {'OTHER_KEY': 'value'};
```

```
// However, object keys are not protected,
```

```
// so the following statement is executed without problem
```

```
MY_OBJECT.key = 'otherValue';
```

```
// The same applies to arrays
```

```
const MY_ARRAY = [];
```

```
// It's possible to push items into the array
```

```
MY_ARRAY.push('A'); // ["A"]
```

```
// However, assigning a new array to the variable throws an error
```

```
MY_ARRAY = ['B'];
```

6.3.5. Expression

Composed of operator and operand, together they form expressions that give a value which should be assigned to an variable, e.g.

`2 + b % c`

2, b, and c are operands; +, % are operators.

Common JavaScript Operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- String Operator
- Conditional Operator

6.3.5.1. Arithmetic Operators

6.3.5.1. Arithmetic Operators

They take numerical values (either literals or variables) as their operands and return a single numerical value.

$x + 2$ // 2 is a literal and x is a variable

Operator	Description	Example		Result
		x	y	
+	Addition	15	5	20
-	Subtraction	15	5	10
*	Multiplication	15	5	75
/	Division	15	5	3
		15	4	3.75
%	Modulus	5	2	1
	(division remainder)	10	8	2
		8	2	0
++	Increment	15		$x++=16$
--	Decrement	15		$x--=14$

6.3.5.2. Special behaviour of String

6.3.5.2. Special behaviour of String

Refer to the following coding:

```
<script>
s1="Hi, I am a CityU student. My student number is:"
s2="20005"
s3=25
</script>
</head>
<body bgcolor="yellow">
<script>
document.write(s1+s2+"<br>")
document.write(s2+ " minus " + s3 + " is: ")
document.write(s2-s3)
</script>
```

Note this special case – **meaning of operator depends on the operands**. Most other programming languages don't have this behaviour.

6.3.5.3. Assignment Operators

An assignment operator assigns a value to its left operand based on the value of its right operand.

<i>Operator</i>	<i>Example</i>	<i>Is the Same As</i>
<code>=</code>	<code>x = y</code>	<code>x = y</code>
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code>
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code>
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code>
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code>
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code>

6.3.5.4. Comparison & Logical Operators

6.3.5.4. Comparison & Logical Operators

A comparison operator compares its operands and returns a Boolean value (true or false). Note the comparison of **both value and type**. The operands can be numerical or string values.

Logical operators are used with expressions that return boolean values.

Operator	Description	Example
<code>==</code>	is equal to	<code>15==5</code> , returns false
<code>===</code>	is equal to (checks for both value and type)	<code>x=15, y="15"</code> <code>x==y</code> , returns true <code>x===y</code> , returns false
<code>!=</code>	is not equal	<code>15!=8</code> , returns true
<code>></code>	is greater than	<code>15>5</code> , return true
<code><</code>	is less than	<code>15<5</code> , return false
<code>>=</code>	is greater than or equal to	<code>15>=5</code> , return true
<code><=</code>	is less than or equal to	<code>15<=5</code> , return false

Operator	Description	Example
<code>&&</code>	and	<code>x=15, y=5</code> <code>(x<20 && y>5)</code> , returns false
<code> </code>	or	<code>x=15, y=5</code> <code>(x<20 y>5)</code> , returns true
<code>!</code>	not	<code>x=15, y=5</code> <code>!(x<20 y>5)</code> , returns false

6.3.5.5. Operator Reference

6.3.5.5. Operator Reference

The reference below is from the book: JavaScript - The Definitive Guide, David Flanagan, O'Reilly, 5th edition

- this is not an easy book to read, intended for programmers

Operator Precedence

- determine the order in which operators are evaluated
- operators with higher precedence are evaluated first
- e.g. $w = x + y * z$

Operator Associativity

- determine the order when operations of the same precedence are performed

- `w = x = y = z;`

Table 5-1. JavaScript operators

P	A	Operator	Operand type(s)	Operation performed
15	L	.	object, identifier	Property access
	L	[]	array, integer	Array index
	L	()	function, arguments	Function call
	R	new	constructor call	Create new object
14	R	++	lvalue	Pre- or post-increment (unary)
	R	--	lvalue	Pre- or post-decrement (unary)
	R	-	number	Unary minus (negation)
	R	+	number	Unary plus (no-op)
	R	~	integer	Bitwise complement (unary)
	R	!	boolean	Logical complement (unary)
	R	delete	lvalue	Undefine a property (unary)
	R	typeof	any	Return datatype (unary)
	R	void	any	Return undefined value (unary)
13	L	*, /, %	numbers	Multiplication, division, remainder
12	L	+, -	numbers	Addition, subtraction
	L	+	strings	String concatenation
11	L	<<	integers	Left shift
	L	>>	integers	Right shift with sign extension
	L	>>>	integers	Right shift with zero extension
10	L	<, <=	numbers or strings	Less than, less than or equal
	L	>, >=	numbers or strings	Greater than, greater than or equal
	L	instanceof	object, constructor	Check object type
	L	in	string, object	Check whether property exists
9	L	==	any	Test for equality
	L	!=	any	Test for inequality
	L	===	any	Test for identity
	L	!==	any	Test for nonidentity
8	L	&	integers	Bitwise AND
7	L	^	integers	Bitwise XOR
6	L		integers	Bitwise OR
5	L	&&	booleans	Logical AND
4	L		booleans	Logical OR
3	R	?:	boolean, any, any	Conditional operator (three operands)
2	R	=	lvalue, any	Assignment
	R	*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, =	lvalue, any	Assignment with operation
1	L	,	any	Multiple evaluation

6.3.6. Flow Control Statements

6.3.6. Flow Control Statements

Common JavaScript Flow Control Statements

- if-else statement
- switch statement
- for statement
- while statement
- do-while statement
- break statement
- continue statement
- return statement
- block statement

6.3.6.1. if-else

6.3.6.1. if-else

Executes a statement (or a block of statements) if a specified condition is true otherwise another statement will be executed.

A single statement

```
if (condition1)
    statement1
else
    statement2
```

To execute multiple statements within a clause, use a block statement { ... } to group those statements.

```
if (condition1) {
    statements1
} else {
    statements2
}
```

```
if (condition1)
    statement1
else if (condition2)
    statement2
else if (condition3)
    statement3
...
else
    statementN
```

6.3.6.2. switch

Too many if-else (nested) considered as bad practice. A multi-branch flow control is easier to follow and less error-prone.

Evaluate an expression, matching the expression's value to a case label; execute statements associated with that case; if no matching label is found, the default clause will be executed.

The optional break statement ensures that the program breaks out of switch once the matched statement is executed giving more efficient codes.

```
switch (expression) {  
    case label1:  
        statements1  
        [break;]  
    case label2:  
        statements2  
        [break;]  
    ...  
    case labelN:  
        statementsN  
        [break;]  
    default:  
        statements_default  
        [break;]  
}
```

6.3.6.3. for loop

Best to understand with an example problem – getting the content of n boxes one by one.

```
for ([initial-expression]; [condition]; [increment-expression])  
{  
  ...  
  statements  
  ...  
}
```

- initial-expression : an expression, including assignment expressions, or variable declaration used to initialize a counter variable (**start with box no. 1, therefore initialize counter to 1**)
- condition : an expression evaluated on each pass through the loop, keep on executing the statements if this condition is true (**keep getting content from boxes if counter is less or equal to n**)
- increment-expression : generally used to increment the counter variable (**move to next box, i.e. add 1 to counter**)

6.3.6.4. while & do-while loop | loops example - <http://courses.cs.cityu.edu.hk/cs2204/example/html/26-DifferentLoops.html>

6.3.6.4. while & do-while loop

While loop

```
while (condition) {  
    statements  
}
```

Similar to for-loop but only use the condition and instead of using counter, the condition will be updated by statements inside the loop. Use the boxes example in previous slide again, the condition is $i < n$. Initialize i before entering into the loop and increment it with one of the statements. Note that **the statements may not be executed at all** if condition is false before entering into the loop.

Do-while loop

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

Similar to while loop but **always do the statements once** first before checking the condition.

6.3.6.5. break, continue, return & block | loops example -

<http://courses.cs.cityu.edu.hk/cs2204/example/html/26-DifferentLoops.html>

6.3.6.5. break, continue, return & block

Break;

terminate the current loop, switch and transfer program control to the statement following the terminated statement

Continue;

terminate execution of the statements in the current iteration of the current loop and go to the next iteration and continue with the loop.

Return;

used inside function, terminate the function execution and may return value to the function caller.

Block - the curly brackets used to group a number of statements into a unit, not actually a statement. Note that Javascript is not a **block structured** language, cannot have { } inside { }.

6.3.7. Debugging

6.3.7. Debugging

Debugging is the process to find and get rid of errors in programs. How to find & fix errors in my Javascript? Nearly all modern browsers provide tools to check the HTML, CSS rules and errors in Javascript, e.g. in Chrome -> options -> more tools -> developer tools. IE, FF and Safari all have similar functions.

The example below has many errors about variables, can try to find them.

Errors in general can be classified into syntax and run-time errors. Syntax errors can be found by your editor tool before running the script. Sometimes, there is no run-time error but the result is not correct. This is called logic error, something wrong with your design or algorithm. In this case, the alert() built-in function will be useful. alert() can be put in different locations in your scripts and give a **trace** of the logic flow.

6.3.8. Javascript Functions

Function is a feature available in nearly all programming languages. It is a "subprogram" that can be **called** by other codes. Commonly used for:

- repeated use of a set of statements
- event handler

There are 2 types of function in Javascript:

- self-defined function - declared by the programmer
- built-in function - defined in Javascript, can be used directly without declaration

6.3.8.1. Function Declaration | **repeated use of codes -**

<http://courses.cs.cityu.edu.hk/cs2204/example/html/t5-echo.html> | **event handler -**

<http://courses.cs.cityu.edu.hk/cs2204/example/html/26-DifferentLoops.html>

6.3.8.1. Function Declaration

A function must be declared before it can be used (called).

```
function name ([param1 [, param2 [, ... paramN]]) {  
    statements  
    [return statement]  
}
```

- name - the function name, should follow the rules for variable declaration
- param - the parameter names used in the function representing the actual value passed in when the function is called (argument); a function can have up to 255 parameters
- statements - refer to the statements comprising the body of the function (the actual work to be done)
- return statement - to specify the value to be returned (if any, the result) from the function
- [] means optional - parameter 1 to N are optional and return is also optional

6.3.8.2. Function Parameter

6.3.8.2. Function Parameter

There is no checking of parameter types

- because Javascript is loosely typed
- you have to check on your own if you want to

Regardless of function declaration

- arguments can be provided, even they are not defined in declaration, when the function is called
- use the arguments object to get the actual arguments
- this is in fact a way to implement function overloading (polymorphism)

```
function f() {  
    for (i=0; i<arguments.length; i++) {  
        alert(arguments[i]);  
    }  
}
```

```
}  
//end of declaration  
f( );  
f(1);  
f("string1", "string2");
```

6.3.8.3. Special Characteristics of Function

6.3.8.3. Special Characteristics of Function

Function declaration can be nested

```
function hypotenuse(a, b) {  
    function square(x) {return x*x;}  
    return Math.sqrt(square(a), square(b));  
}
```

Function is an object and therefore can be assigned to variable

```
function square(x) {return x*x;}  
var a = square(4);  
var b = square;  
var c = b(5);
```

Function can have no name - anonymous function

```
var d = function(x) {return x*x;}  
var e = d(3);
```

The latter two characteristics are commonly used in event handler or object method set up.

6.3.8.4. Built-in Javascript Function

6.3.8.4. Built-in Javascript Function

Function	Description
<u>decodeURI()</u>	<i>Decodes an encoded URI</i>
<u>encodeURIComponent()</u>	<i>Encodes a string as a URI</i>
<u>escape()</u>	<i>Encodes a string</i>
<u>eval()</u>	<i>Evaluates a string and executes it as if it was script code</i>
<u>isFinite()</u>	<i>Checks if a value is a finite number</i>
<u>isNaN()</u>	<i>Checks if a value is not a number</i>
<u>Number()</u>	<i>Converts an object's value to a number</i>
<u>parseFloat()</u>	<i>Parses a string and returns a floating point number</i>
<u>parseInt()</u>	<i>Parses a string and returns an integer</i>
<u>String()</u>	<i>Converts an object's value to a string</i>

6.4. Conclusions | **Example - <http://courses.cs.cityu.edu.hk/cs2204/example/html/28-BuiltinFunctions.html>** |
Example from w3school - http://www.w3schools.com/jsref/jsref_obj_global.asp

6.4. Conclusions

Extension readings:

- Text book chapter 11
- Javascript guide from Mozilla/
- <http://www.w3schools.com/jsref/>

6. Javascript Part I | MDN - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction> | w3schools - <http://www.w3schools.com/jsref/>