SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10011/60004
Creating Web Applications

Lecture 6: JavaScript Part 2

- JavaScript objects
- Client Data Validation
- Program Flow Control

Last Lecture - JavaScript syntax +

HTML - *content*

```
<!DOCTYPE html>
<html lang="en">
<head>
    …
    <script src="my_jsfile.js"></script>
</head>
<body>
    ...
    <p><span id="mymessage"></span></p>
    <p><button type="button" id="clickme">
    Click Me!</button></p>>
    ...
</body>
</html>
```

JavaScript - *behaviour*

```
/* Filename: my_jsfile.js
   …
*/

function doSomething()
{
      var myString, outputMessage;     //declare local variables
      myString = prompt("Enter the string", "The string");
      alert("Your output: " + myString);
      outputMessage = document.getElementById("mymessage");
      outputMessage.textContent="Your output: " + myString;
}

function init() {
      var clickme = document.getElementById("clickme");
      clickme.onclick = doSomething;
      }

window.onload = init;
```

Previously – JavaScript Syntax

- Statements
- Data Types
  - Primitive
- Variables
  - Naming
  - Variable scope
- Constants
- Expressions
  - Operators
    - String, Arithmetic, Logical, Comparison
    - Assignment
- Functions
  - function definition
  - parameters
  - call and return

This lecture …

- JavaScript objects
- In-built JavaScript objects and functions
  - Array, Date, String
  - Global functions
- Validating Form Data using JavaScript
  - Regular expressions revisited
- Debugging JavaScript
  - Web Developer/Debugger
    Breakpoints, Watch variables
- Flow control in JavaScript (reference slides)
  - Sequence
  - Selection
  - Repetition

Focus on what is different from the languages you already know

SWINBURNE UNIVERSITY OF TECHNOLOGY

?

- Where have we seen *object.property* and *object.method* notation in JS before?

```
function doSomething()
{
      var myString, outputMessage;     //declare local variables
      myString = prompt("Enter the string", "The string");
      alert("Your output: " + myString);
      var outputMessage = document.getElementById("mymessage");
      outputMessage.textContent="Your output: " + myString;
}

function init() {
      var clickme = document.getElementById("clickme");
      clickme.onclick = doSomething;
      }

window.onload = init;
```

object property

object method

Custom Objects – Make Your Own

Create a specific instance of a **ball** object from class **Ball()** placed it in an object 'container' named **myObject**

- To create

  `myObject = new Ball(…)`

- To access the ball's **properties**

  `myObject.colour`

  `myObject.size`

- To access the ball's **methods**

  `myObject.bounce()`

  `myObject.roll()`

In this unit we will use pre-defined objects.

- ***JavaScript core objects***
- ***Browser / DOM objects***

In more advanced programming you will create you own objects.

This lecture …

- JavaScript objects
- In-built JavaScript objects and functions
  - String, Array, Date, RegExp
  - Global functions
- Validating Form Data using JavaScript
  - Regular expressions revisited
- Debugging JavaScript
  - Web Developer/Debugger, Breakpoints, Watch variables
- Flow control in JavaScript
  - Sequence
  - Selection
  - Repetition

Predefined Objects – JS Core Objects

- **String**
- **Array**
- Boolean
- **Date**
- Math
- Number
- **RegExp**

Example

```
// returns PI
var x = Math.PI;
```

Object prototype
Note: starts with a Capital Letter

https://developer.mozilla.org/en/JavaScript/Guide/Predefined_Core_Objects

String Object

- Is a *wrapper* for a **string** primitive
- Properties - length
- Methods

charAt()	Returns the character at the specified index (position)
match()	Searches a string for a match against a regular expression, and returns the matches
replace()	Searches a string for a value and returns a new string with the value replaced
search()	Searches a string for a value and returns the position of the match
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts a part of a string from a specified position
toLowerCase()	Converts a string to lowercase letters

Example String method

```
//charAt(position)
var message="jquery4u"
alert(message.charAt(1)) //alerts "q"

//replace(substr, replacetext)
var myString = '999 JavaScript Coders';
console.log(myString.replace(/Javascript/i, "jQuery"));
//output: 999 jQuery Coders

//slice(start, end)
var text="excellent"
text.slice(0,4) //returns "exce"
text.slice(2,4) //returns "ce"
```

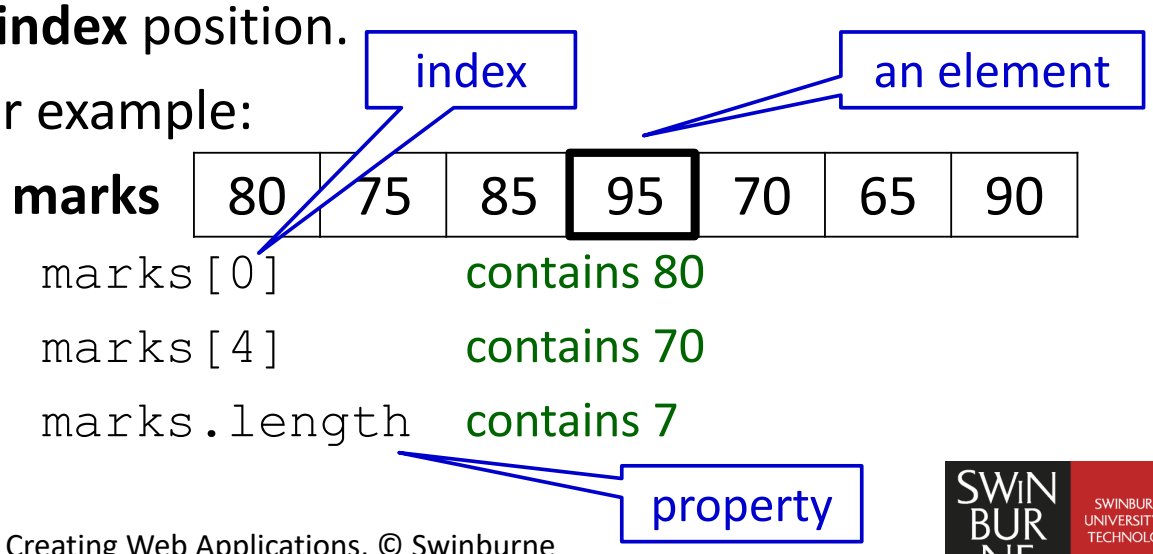
Regular Expression
What's /i ?

Up to, but not including end

Array Object

- an indexed collection of variables
- a particular location or **element** in an array is referenced by the name of the array and the **index** position.

For example:



Where have we seen arrays in html?

```
<fieldset id="categories">
    <legend>Competition Categories</legend>
    <p>Select which categories your would like your cat entered</p>
    <p><label for="bestbreed">Best of Breed (adult)</label>
        <input type="checkbox" id="bestbreed" name="categories[] " value="best"/>
    </p>
    <p><label for="kit">Best of Breed (kitten)</label>
        <input type="checkbox" id="kit" name="categories[]" value="kitten"/>
    </p>
    <p><label for="mog">Best Non-Pedigree</label>
        <input type="checkbox" id="mog" name="categories[]" value="moggy"/>
    </p>
</fieldset>
```


Array Object (continued)

- In JavaScript an **Array** is an object.
- An (empty) array can be created as follows:

```
var marks = [];
```

square brackets

- Alternatively, the **new** keyword can be used to create an instance of an Array object of given length.

```
var marks = new Array(15);
```

parenthesis

use plural form to indicate array

15 places, 0-14

Array Object (continued)

- Element values may be set in an **initialiser list**:

```
var subjects = ["CWA","WAD","WAA"];
```

better

same as `var subjects = new Array("CWA","WAD","WAA");`

```
var numbers = [1,1,2,3,5,8,13];
```

- Alternatively values may be set after the array has been allocated by referring to the index position of the particular array element:

```
var favSubjects =[];
favSubjects[0] = "CWA";
favSubjects[1] = "WAD";
```

create empty array

dynamically extend

Array Object (continued)

- The length of an array can be accessed using the **`length`** property, e.g. **`numbers.length`**
- Values can be set programmatically:

```
// create an array
var numbers = new Array(100)
// fill array with numbers
for (var i=0; i < numbers.length; i++) {
    numbers[i] = i*2;
}
// display the last element
alert (numbers[numbers.length - 1]);
```

Why not subtract 1?

Why subtract 1?

Array Object (continued)

- There are several **predefined** arrays in the **document** object, such as link**s**, frame**s**, image**s**

```
myLink = document.links[0];
myImage = document.images[5];
myNode = document.documentElement.children[0];
```

Pre-defined arrays uses plural form to indicate collection of elements

Array Object - Example

Example: Display scores array as a horizontal 'chart'

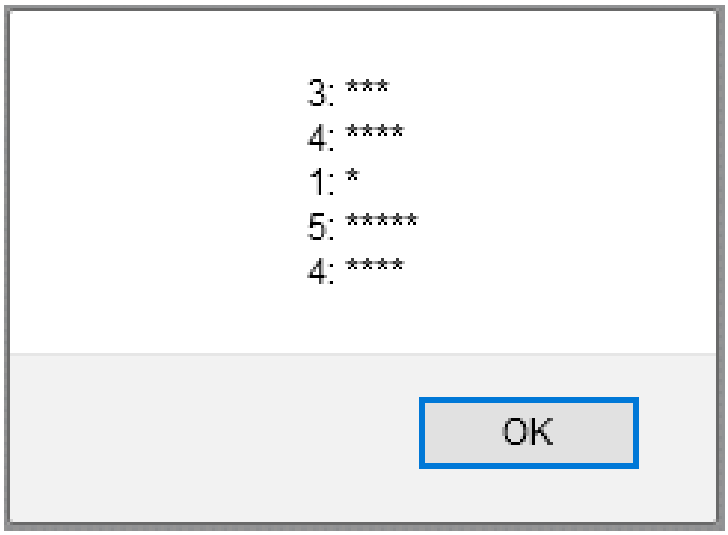
```
var scores = [3,4,1,5,4];
var num;            // current number
var ans = "";       // string for temp output
var msg = "";       // string for final output
// Demonstrates how to use for-in loop
for (var i=0; i < scores.length; i++) {
      num = scores[i];
      ans = num.toString() + ": ";
      for (var j=0; j<num; j++) {
            ans = ans + "*";
      }
      msg = msg + ans + "\n";
} // external for loop
alert (msg);
```

Method to convert a number to a string

\n for line break

Array Object - Example (continued)

Example: alert box will display



Array Object – Properties/Methods

Property/Method	Description
`length`	returns length of the array
`join(delimiter)`	makes a string delimited with the items
`pop()`	removes the last and return it
`push(item)`	Add item to end
`reverse()`	reverses the order of items
`shift()`	removes first item and returns it
`slice(start,[end])`	returns a sub-array
`sort(fn)`	fn needs (a<b)==-1, (a==b)=0, (a>b)==1
`unshift(item)`	add item to start of array

https://developer.mozilla.org/en/JavaScript/Guide/Predefined_Core_Objects

Date Object

- Represents a date that allows computation
- Numeric value is expressed as millisecond

```
var d = new Date("May 8, 2018 17:30:00");
```

"Constructor method"
Full or 3-letter month

- `var d = new Date();`

New instance of **client's** current date and time

- **Methods** can be used to obtain values within the date object

```
var n = d.getDate();
```

Date Object - Some Methods

Method	Description
`getDate()`	Returns the day of the month (from 1-31)
`getDay()`	Returns the day of the week (from 0-6)
`getFullYear()`	Returns the year (four digits)
`getHours()`	Returns the hour (from 0-23)
`getMilliseconds()`	Returns the milliseconds (from 0-999)
`getMinutes()`	Returns the minutes (from 0-59)
`getMonth()`	Returns the month (from 0-11)
`getSeconds()`	Returns the seconds (from 0-59)

From our demo ....

```
function getAgeInYears(){
    var age = -1;       //can be used to check if error
   // store number of millisecs in a non-leap year as a constant
   const YEAR_IN_MILLISECS = 365 * 24 * 60 * 60 * 1000;
    // get the current date-time
   var now = new Date();
   //get dob as string
   var dobStr = document.getElementById("dob").value;
   //split date into array with elements dd mm yyy
   var dmy = dobStr.split("/");
   var dob = new Date(dmy[2],dmy[1],dmy[0],0,0,0,0);
   //constructor parameters (year, mth, day, hrs, mins, seconds, ms)
   return age = (now.valueOf() - dob.valueOf())/YEAR_IN_MILLISECS;
   //time is calculated in milliseconds
}
```

String method that returns an Array

This lecture …

- JavaScript objects
- In-built JavaScript objects and functions
  - Array, Date, String
  - Global functions
- Validating Form Data using JavaScript
  - Regular expressions revisited
- Debugging JavaScript
  - Web Developer/Debugger
    Breakpoints, Watch variables
- Flow control in JavaScript (reference slides)
  - Sequence
  - Selection
  - Repetition

JavaScript - Global Functions

Be careful of case

Function	Description
eval()	Evaluates a string and executes it as if it was script code
isFinite()	Determines whether a value is a finite, legal number
isNaN()	Determines whether a value is an illegal number
Number()	Converts an object's value to a number
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer
String()	Converts an object's value to a string

JavaScript - Global Functions (examples)

Function	Example	Result
eval()	eval("2 + 3")	5
isFinite()	isFinite(5) isFinite("Web")	true false
isNaN()	isNaN(5) isNaN("Web")	false true
Number()	Number("22") Number("2 2")	22 NaN
parseFloat()	parseFloat("2") parseFloat("2.34") parseFloat("2 34") parseFloat("2 units") parseFloat("unit 2")	2 2.34 2 2 NaN

From our demo ....

```
function isDobOK(){
    var validDOB = true;            //set to false if not ok
    var now = new Date();           //current date-time
    var dob = document.getElementById("dob").value;
    var dateMsg = "";
    //split date into array with elements dd mm yyyy using / as a separator
    var dmy = dob.split("/");
    for (var i = 0; i < dmy.length); i++){
        if(isNaN(dmy[i])){        //for each part of date check is number
            dateMsg = dateMsg
                    + "You must enter only numbers into the date" + "\n";
            validDOB = false;
        }
    }
    ...
```

JavaScript - Global Functions (examples)

Function	Example	Result
parseInt()	parseInt("2") parseInt("2.34") parseInt("2 34") parseInt("2 units") parseInt("unit 2")	2 2 2 2 NaN
String()	String (0) String (true) String ("2")	0 true 2

" are not displayed

This lecture …

- JavaScript objects
- In-built JavaScript objects and functions
  - Array, Date, String
  - Global functions
- Validating Form Data using JavaScript
  - Regular expressions revisited
- Debugging JavaScript
  - Web Developer/Debugger
    Breakpoints, Watch variables
- Flow control in JavaScript
  - Sequence
  - Selection
  - Repetition

RegExp in JavaScript – Example 1

- Simple check for a phone number using the String object method **match()**

Notice:
/.../ not "..."

Need ^ ... $ if you want to match the whole string (unlike HTML5)

```
function checkPhoneNumber(phoneNo) {
    var phoneRE = /^\(\d\d\) \d\d\d\d-\d\d\d\d$/;
    var isOk = false;
    if (phoneNo.match(phoneRE)) {
        isOk = true;
    } else {
        alert( "The phone number entered is invalid!" );
    }
    return isOk;
}
```


RegExp in JavaScript – Example 2

- Simple check for a phone number using the RegExp object method **test()**

```
function checkPhoneNumber(phoneNo) {
    var phoneRE = /^\(\d\d\) \d\d\d\d-\d\d\d\d$/;
    var isOk = false;
    if (phoneRE.test(phoneNo)) {
        isOk = true;
    } else {
        alert( "The phone number entered is invalid!" );
    }
    return isOk;
}
```

RegExp in JavaScript

- Initialise a Regular Expression -

```
re = /bana+na/;      // or
re = new RegExp("bana+na");
```

- String methods **`match()`** `replace()` `search()` `split()`

```
str = "For more information, see Chapter 3.4.5.1";
re = /(chapter \d+(\.\d)*)/i;
found = str.match(re); // returns true or false
```

i modifies the search to be case-insensitive

- RegExp methods **`test()`** `exec()`

```
found = re.test(str); // returns true or false
```


From our demo ….

```
function chkEmail () {
    var email = document.getElementById("email");
    var result = false;
    var pattern =
        /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-za-zA-Z0-9.-]{1,4}$/;
    if (pattern.test(email.value))  {
          result = true;
    }
    else  {
        result = false;
        gErrorMsg = gErrorMsg + "Enter a valid email address\n"
    }  // else
    return result;
}
```


Forms and JavaScript

- JavaScript provides much greater **control** over the use of forms by:
  - **Checking form values** entered, before the form is submitted:
    - check that **required** form values have been supplied
    - check that values **conform to a type**
      (eg, must be an integer, or a string, etc)
    - check that values are **logical** or **constrained**
      (eg. end date after start date, value in a range, etc)
  - **Alerting users** if inappropriate form values have been entered
  - **Reassuring users** that their form input has been successfully processed and transmitted
  - **Adaptively presenting** new forms to a user, based on user's responses to prior forms.
  - **Pre-processing** form data before submission

Using HTML5 to control / check data input

- HTML Forms Input Data Control and Checking

```
<form id="regForm" method="post“ action="….php">
    …
    <p><label for="catname">Cat&amp;apos;s Name</label>
    <input type="text" name= "catname" id="catname"
          maxlength="20"
          size="10"
          required="required"/>
    </p>
    …
    <input type="email" name= "email" id="email" required="required"/>
</form>
```

From Lecture 2

Restricts the # characters

Size of text box

A value must be entered

HTML5 input control

From Lect 2 - Using patterns in HTML

- The pattern attribute uses a 'regular expression' to define the characters that can be entered into a field

```
<input type="text" name= "catname" id="catname"   maxlength="20"
       pattern="^[a-zA-Z ]+$"
       required="required"/>
```

From Lecture 2

Alpha characters or space only

```
<input type="text" name= "dob" id="dob"  maxlength="10" size="10"
       placeholder="dd/mm/yyyy"
       pattern="\d{1,2}/\d{1,2}/\d{4}"
       required="required"/>
```

Placeholders provide prompt to the user

dd/mm/yyyy
???? no range

```
/(?=d)(?:(?:31(?!.(?:0?[2469]|11))|(?: 30|29)(?!.0?2)|29(?=.0?2.(?:(?:(?:1[6--9]| [2--9]d)?(?:0[48]|[2468][048]|[13579]
[26])|(?:(?:16|[2468][048]|[3579] [26])00)))(?:x20|$))|(?:2[0--8]|1d|0? [1--9]))([--/])(?:1[012]|0?[1--9])1(?: 1[6--9]|[2-
-9]d)?dd(?:(?=x20d)x20| $))?/
```

Regular expressions not necessarily the best solution to every check!!

http://html5pattern.com/

Checking Form Data with JavaScript

Given the following HTML form, *take note of the* ***IDs***

```
<form id="regform" method="post"
   action="process.php">
   <div class="textinput">
      <label for="firstname">First Name</label>
      <input type="text" name="firstname"
                              id="firstname" />
   </div>
   <div class="textinput">
      <label for="age">Age</label>
      <input type="text" name="age" id="age" />
   </div>
   <div class="buttoninput">
      <input type="submit" value="Register" />
   </div>
</form>
```

submit will **action** **form**

Checking Form Data - Template

```
function validateForm() {
  /* code here */
  return result;
}
function initialise() {
 var formElement =
   document.getElementById("regform");
 formElement.onsubmit = validateForm;
}

window.onload = initialise;
```

Write the data validation code, and return **true** if valid, otherwise **false**.
*Form will not be **actioned** if **onsubmit** is **false***

Form ID in HTML

Register a function to respond to the submission of the form

Checking Form Data - Example (cont)

```
function validate() {
 var errMsg = "";
 var result = true;
 var firstName =
 document.getElementById("firstname").value;
 var age =
     document.getElementById("age").value;
```

Initialise some local control variables

value property of an HTML form control element

Local variables used, as they are only used within the function

Checking Form Data - Example (cont)

```
if (firstName == "") {
   errMsg += "First Name cannot be empty.\n";
}
if (age == "") {
    errMsg += "Age cannot be empty.\n";
}
if (isNaN(age)) {
    errMsg += "Age is not a valid number.\n";
}
```

Add a new line for when errMsg displayed in the alert box

Concatenate error message

Use global function to check if age contains a valid number

Checking Form Data - Example (cont)

```
if (errMsg != "") {
    alert (errMsg);
    result = false;
  }
  return result;
}
```

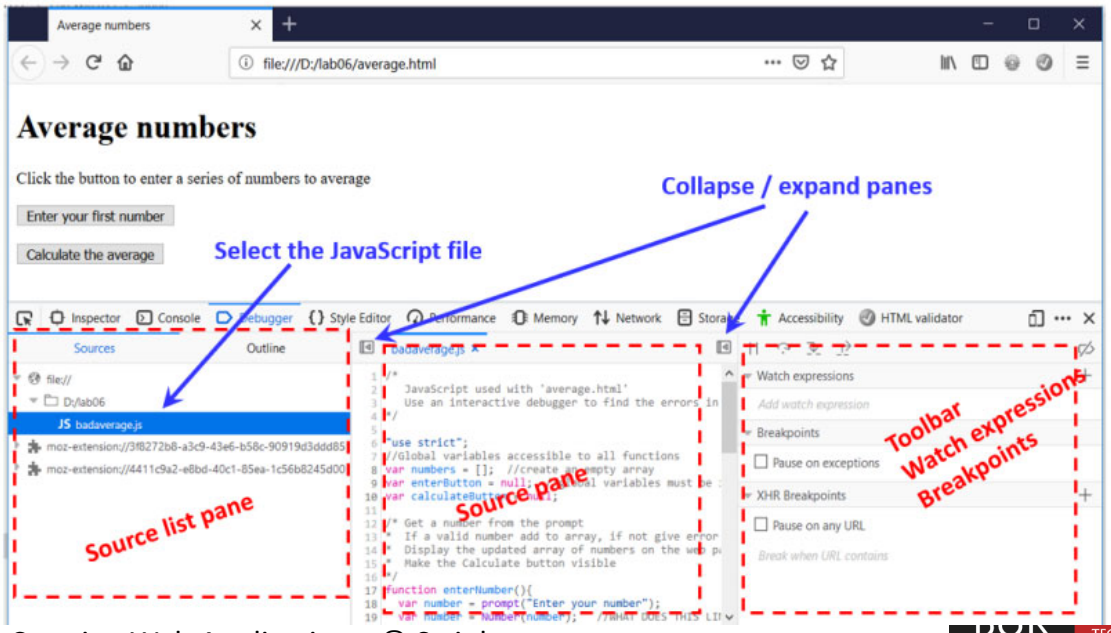
Checks if any error messages concatenated

Error detected

Returns **true** (set earlier) if no errors detected, otherwise **false**

Debugging JavaScript

- Web Developer/Debugger
  See the Lab exercise….

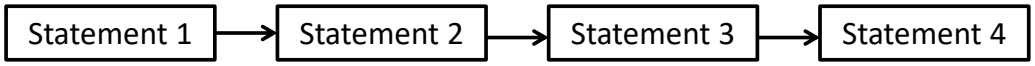


Reference slides

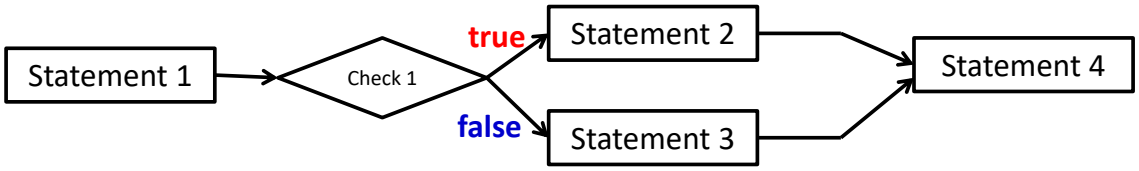
- JavaScript objects
- In-built JavaScript objects and functions
  - Array, Date, String
  - Global functions
- Validating Form Data using JavaScript
  - Regular expressions revisited
- Debugging JavaScript
  - Web Developer/Debugger
    Breakpoints, Watch variables
- Flow control in JavaScript
  - Sequence
  - Selection
  - Repetition

Three Models in Programming

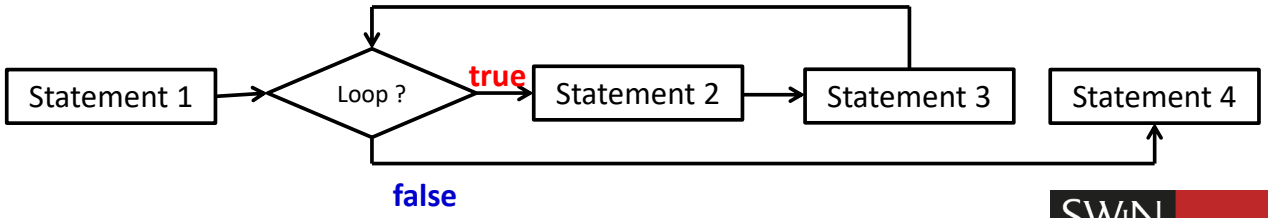
Sequence



Selection

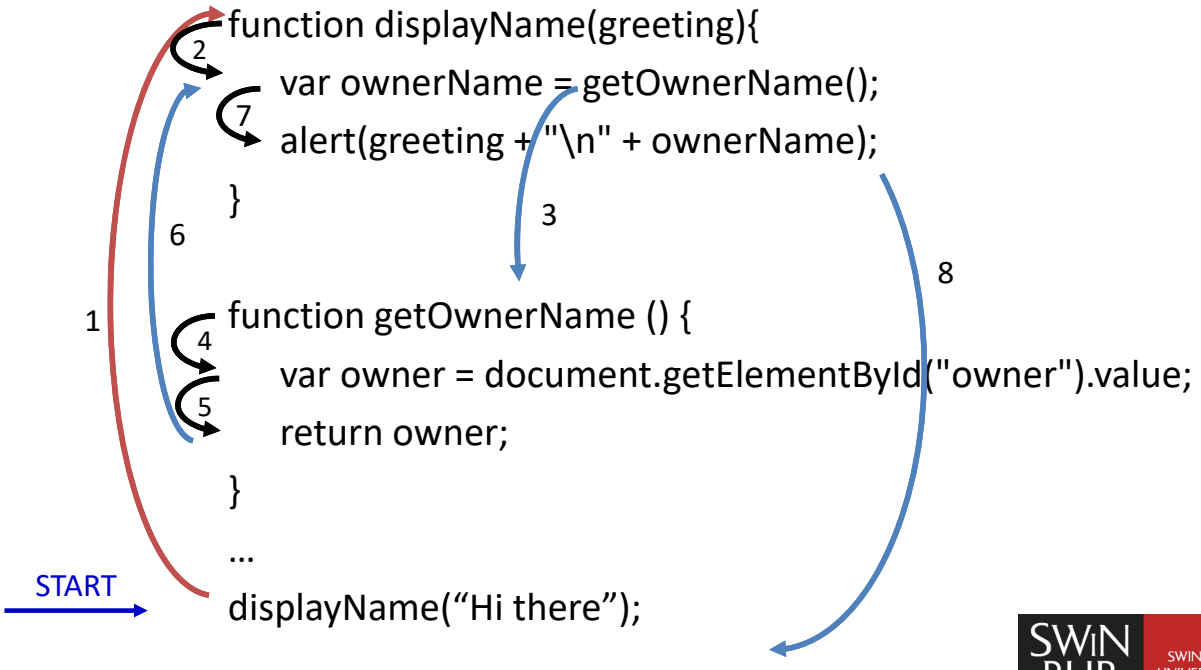


Repetition



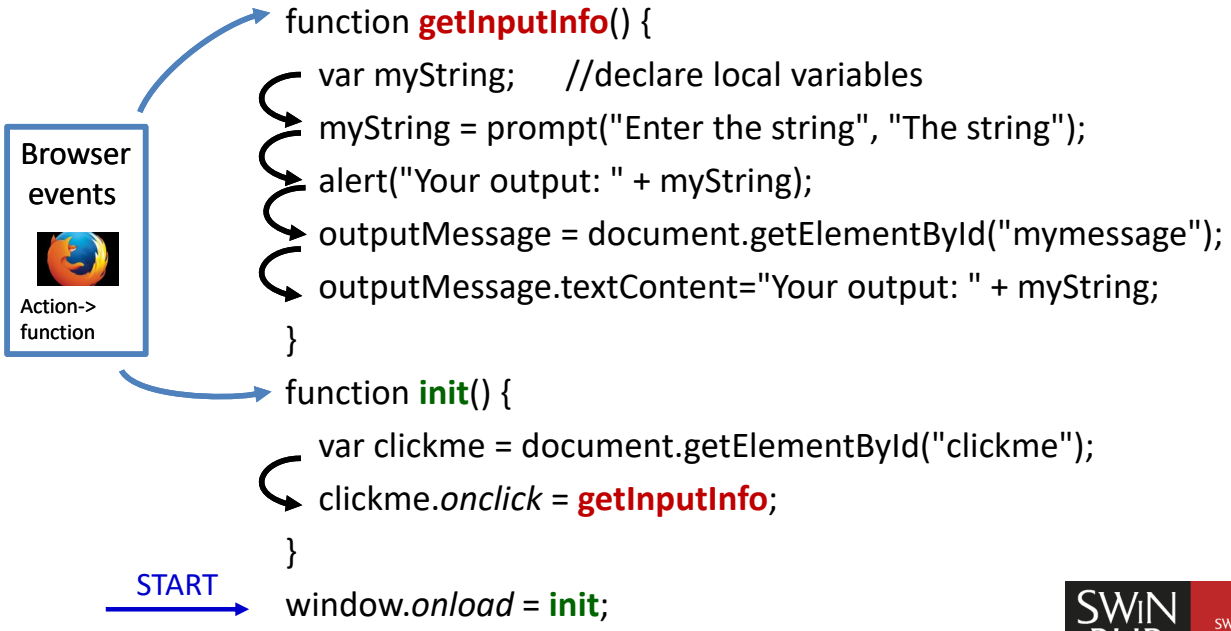
Sequence – Example with function calls

- Where does the sequence of execution start?



Sequence – Event-driven example

Where does the sequence of execution start?



Sequence

- Statements are executed in sequence, one line at a time
- Functions are not executed, unless called
- When functions are called, execution jumps into the function and continues to the next line after the call, once the function completes its execution.
- In an event-driven JS program functions can be called in response to browser actions/events

Selection

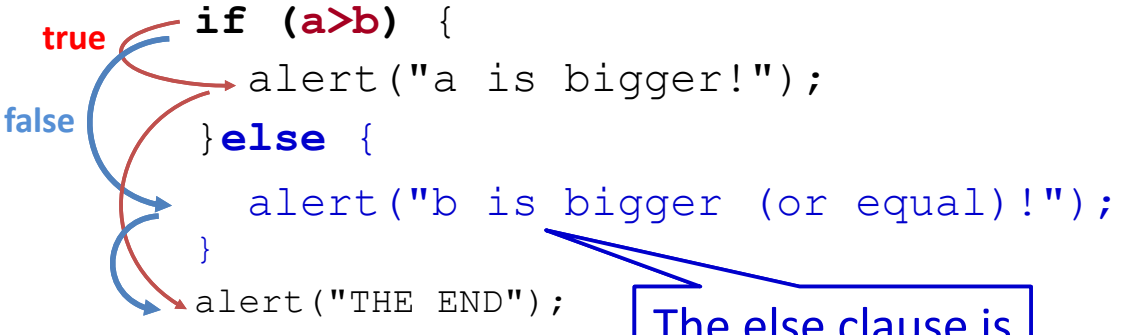
- **Selection,** also referred to as **decision making** or **flow control,** is the process of determining the order in which statements execute in a program
- Statements used for making decisions are called **selection** or **decision-making statements** or **decision-making structures**
- There are 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.



DEMO!

Selection – `if` statement (continued)

- ***condition*** can be any expression that evaluates to **true** or **false**.
- If ***condition*** evaluates to **true**, `statement_1` is executed; otherwise `statement_2` is executed.
  In this case, the sequence on execution is changed.
- `statement_1` and `statement_2` can be *any* statement, including further nested **`if`** statements.

Selection – `if` statement (continued)

- **Example #1**

```
var guess;
var secret = 10;
guess = prompt("Enter a number:");
if (guess == secret){
   alert("Correct number: "+secret);
} else {
   alert("Wrong number");
}
```

== means equals

Selection – `if` statement (continued)

- **Example #2**

```
var guess, msg;
var secret = 10;
guess = prompt("Enter a number:");
if (guess == secret){
  msg = "Correct number: "+secret;
} else {
  msg = "Wrong number";
}
alert(msg);
```


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)

Example #3

```
var guess, msg;
var secret = 10;
guess = prompt("Enter a number:");
if (guess == secret){
  msg = "Correct number: "+secret;
} else if (guess > secret){
  msg = "Number too high";
} else {
  msg = "Number too low";
}
alert(msg);
```


Selection – `if` statement (continued)

Example #4

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

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

Selection – `if` statement (continued)

Example #5

Useful model for a javascript used by many pages

```
function init(){
 if(document.getElementById("news")!==null){
   loadNews();
   var newsForm = document.getElementById("news");
   newsForm.onsubmit=validate_news;
 }
 if(document.getElementById("login")!==null){
   var loginForm = document.getElementById("login");
   loginForm.onsubmit=validate_login;
 }
}

window.onload = init;
```


Selection – `switch` statement

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

Selection – **switch** statement (cont)

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

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


Selection – **switch** statement (cont)

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

Optional

Optional clause

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 continues execution at the statement following the end of switch.

Selection – `switch` statement (cont)

- By convention, the **default** clause is the last clause, but it does not need to be so.
- 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 execution at the statement *following* the switch.
- If **break** is omitted, the program continues execution at the next statement *in the switch* statement.

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 break 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 series of statements while a specific condition is *true* or until a specific condition becomes *true*
- There are four types of loop statements:

  `for` statements

  `for-in` statements

  `while` statements

  `do-while` statements

Repetition – `for` statement

- **Example**

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

initialise

continuation condition

update

DEMO!

- *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)

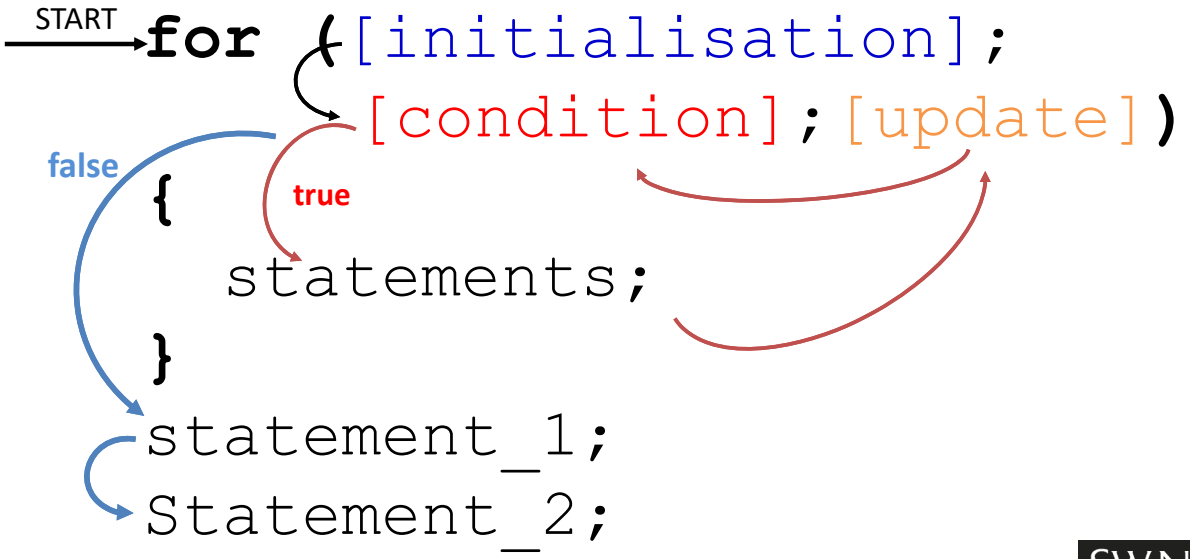
- The `condition` represents a check that determines if the loop is repeated.
  - If the `condition` is **true**, the code block in the loop is executed.
  - If the `condition` is false, the `for` loop terminates.

```
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.



Repetition – `for-in` statement

- The for…in loop is a special loop that allows easy iteration through a group of properties of an object.
- JavaScript provides a number of built in collections which are often used suitable for use with a for...in loop.

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

Repetition – `for-in` statement (cont)

- **Example**

```
var allUnits =
   ["Creating Web Applications",
    "Web Application Development"];
var oneUnit;
var ans = "";
for (oneUnit in allUnits) {
   ans = ans + allUnits[oneUnit]);
}
alert(ans);
```

Declare as an array

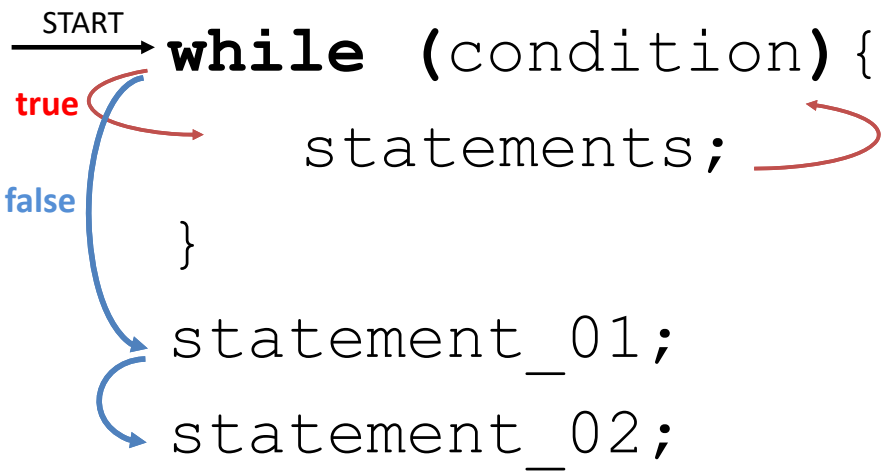
Collections will be discussion in the next lecture

- *What will be displayed?*

Repetition – `while` statement

- A **while** loop uses a *pre-loop test*.
  This means there is a possibility that the statements might never be executed
- To execute multiple statements, use a block statement { ... } to group those statements.
- If the **condition** becomes **false**, statement within the loop stops executing and control passes to the statement following the loop.

Repetition – **while** statement (cont)



Repetition – **while** statement (cont)

- **Example**

```
var i = 0;
var sum = 0;
while (i < 3){
   i = i + 1;
   sum = sum + i;
}
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 pass: i = 1 and sum = 1
After the second pass: i = 2 and sum = 3
After the third pass: i = 3 and sum = 6

After completing the third pass, the condition `i < 3` is no longer true, so the loop terminates.

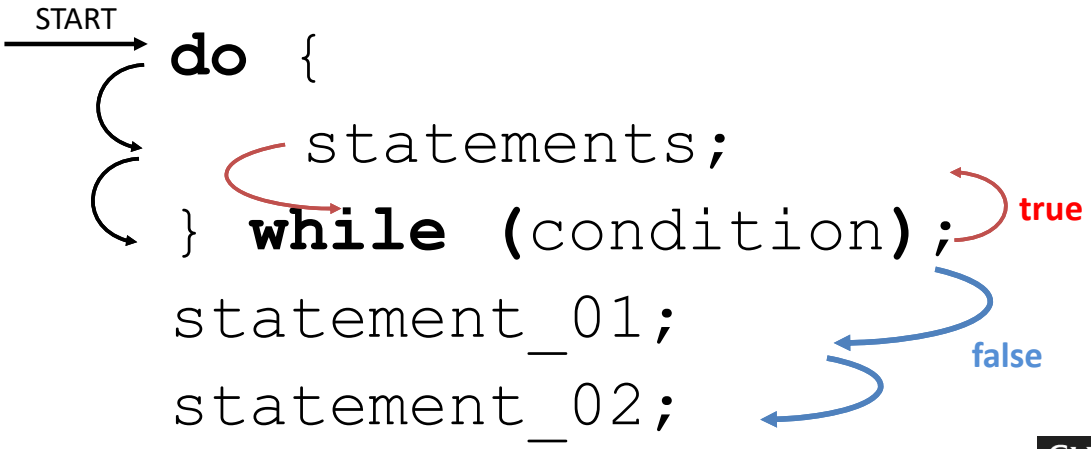
- *What will be displayed?*

Repetition – **do…while** statement

- A **do-while** loop uses a *post-loop test*, which means statements will be executed at least once
- To execute multiple statements, use a block statement { ... } to group those statements.
- If **condition** is **true**, the statement executes 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`.



Repetition – do...while statement (cont)

- **Example #1**

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

What will be displayed?

Repetition – do...while statement (cont)

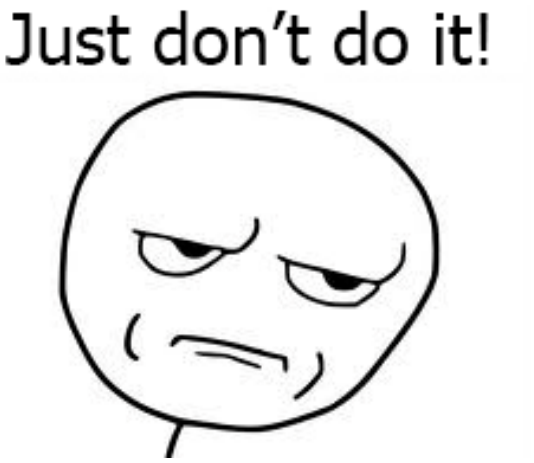
Example #2

```
var guess, msg;
var stop="n";
var secret = 10;
do {
 guess = prompt("Enter a number:");
 if (guess == secret){
   msg = "Correct number: "+secret;
   stop="y";
 } else if (guess > secret){
   msg = "Number too high";
 } else {
   msg = "Number too low";
 }
 alert(msg);
} while (stop=="n");
```

Instead of hard coding 'secret', change the line to:
secret= Math.floor(Math.random() * (100 - 1)) + 1;

Repetition (continued)

- The normal flow of the loop can be altered with the use of
  - Break + label
  - Continue + label
- **However, as a good programming practice *these should be avoided*, and hence will not be covered here**



From our demo ....

HTML DOM element method that returns an array of HTML elements with the parameter name

Array variable

Array property giving the number of array elements

Array element

```
function isCategorySelected(){
    ...
    var categories =
      document.getElementById("categories").getElementsByTagName("input");
    var labels =
      document.getElementById("categories").getElementsByTagName("label");
    var label = "";
    var catList = "";
    for (i=0; i<categories.length; i++){          //for each category element
        selected = selected ||  categories[i].checked;   //see if it is checked
        label = labels[i].firstChild.textContent;        //get its label
        catList = catList + label + "\n";
    }
    ...
}
```

categories and **labels** are *parallel* arrays

Next Lecture

What's Next?

- Document Object Model
- Internet and Web Protocols

SWINBURNE UNIVERSITY OF TECHNOLOGY