EECE1012
Net-Centric Introduction to Computing
JS Review

Amirhossein Chinaei

Winter 2019

Office Hours: T 9:30-10:30 W 11:30-12:30 LAS3048

ahchinaei@cse.yorku.ca

JS variables

- variables are used to store and retrieved data.
- variables are defined by the keyword **var**
- variables are categorized into different **data types**
- first character must be a letter or an underscore (_)
- the rest of the variable can be any letter, number, or underscore
- variable names are case sensitive
  - age, Age, AGE would all be different variable names
- You cannot use JS reserved words for variable names

variable name examples

- valid names:

```
_myVar     thissisalongvariablename     num
_var       eecs1012                     myString
name1      test_1                       x
```

- invalid names:

```
1test          /* starts with a number */
test 1         /* there is a space in the name */
t$est          /* non alphanumeric character */
var            /* reserved word */
```

JS data types

TYPE	Explanation	Example
`Number`	Integers and numbers with decimal places.	99, 2.8, 5, -10
`String`	A variable that can hold a collection of characters. Sometimes we call this a string literal.	"Hello", "EECS1012"
`Boolean`	A variable that holds only two possible values – **true** or **false**.	true or false
`undefined`	When a variable does not have a value	var x;
`Objects`	Objects are special data types that have functions and data associated with them. These are more common in JS than PHP and we will need to use them often.	document.getElementById(); (example of an object)

`function`	A user defined function that can be called by a user event (e.g. mouse click, etc)	`function name ()`
		`{`
		`statements;`
		`...`
		`}`

number type variables

```
var enrollment = -99;
var medianGrade = 70.8;
var credits = 5 + 4 + (2 * 3);
```

- Number types are integers (whole numbers) and numbers with decimal places
- Numbers with decimal places are often called "floating point" numbers, e.g.:

  2.99993   3000.9999   -40.00

  We call them floating point because the decimal point appears to float around. Sometimes these are just called *floats* to distinguish them from integers.

expressions and statements

- An expression is the combination of one or more variables, values, operators, or functions that computes a result.

```
var num1 = 5;             /* value 5 is the expression */

var num2 = num1 + 10;      /* num1 + 10 is the expression,
                           /* this computes 5 + 10 */

num2 = num2 + 1;          /* this uses num2 and assigns the
                             result back to num2 */

var str1 = "hello";                  /* value is "hello" */
var str2 = "world";                  /* value is "world" */
num1  = ((3.14) * 10.0) / 180.0;     /* multiple operators */
```

basic arithmetic operators

a + b	Addition	Sum of a and b.
a - b	Subtraction	Difference of a and b.
a * b	Multiplication	Product of a and b.
a / b	Division	Quotient of a and b.
a % b	Modulo	Remainder of a divided by b.

Here a and b could be variables, but we could also replace them with numbers.   10 + 20, 3.14 / 2.0, etc. . .

“short hand” assignment operators

Assignment	Same as:	
`a += b;`	`a = a + b;`	Addition
`a -= b;`	`a = a - b;`	Subtraction
`a *= b;`	`a = a * b;`	Multiplication
`a /= b;`	`a = a / b;`	Division
`a %= b;`	`a = a % b;`	Modulus
`a++;`	`a = a + 1;`	Self Addition
`a--;`	`a = a -1;`	Self subtraction

JS math operator precedence

```
var num1 = 5 * 5 + 4 + 1 / 2;     /* What is the answer? */
var num2 = 5 * (5 + 4) + 1 / 2;   /* What is the answer? */
                                                       JS
```

```
29.5 and 45.5
                                                   output
```

Operator Precedence:

( )	Highest
* / %	
+ -	Lowest

* This operator precedence is the same for most programming languages.

string type variables

- strings are treated like a series of characters

```
var favoriteFood = "falafel";

var stringNumber = "234";
```

Here, variable `stringNumber` is not the value two hundred and thirty four, but instead the characters 2,3,4.

```
var len = stringNumber.length; /* len is assigned the number 3 */
```

- string variables have a special property called "length" that returns the number of characters in the string.

- keep in mind that spaces are also characters.
- `var stringNumber = "2 34";`

string as an object

```
var s1 = "Connie Client";
var len = s1.length;
```

variable s1 is of
type String, however,
this can also be thought
of as a "String object".

we can access various properties
of the object using a "." operator and
object's the property's name.

You see this type of property access often in
JS (and other "Object Oriented" languages)

more on strings

- You need to put quotes around a string to let JS know it is a string. **You can also use single quotes**.

```
var s = "EECS1012";   // CORRECT!
var s = 'EECS1012';   // CORRECT!


var s = EECS1012;       // INCORRECT! ✗
```

In this example, JS will interpret EECS1012
as a variable, not a string!

special characters

- what if you want a quote character " to be part of the string?

```
var s = "This is a quote " "; //INCORRECT! ✗
```

this statement will cause problems for JS, because when it sees the second double quote it will assume this is the end of the string.

escape characters

- Escape characters are used inside strings to tell JS how to interpret certain characters

```
var s = "This is a quote \" "; //CORRECT
```

This string will be interpreted in JavaScript as:
T-h-i-s-_-i-s-_-a-_-q-u-o-t-e-_-"-_

Here a – is used to separated characters.
An underscore _ is used to represent a space character.

alternatives

```
var answer = "It's alright";
var answer = "He is called 'Johnny'";
var answer = 'He is called "Johnny"';
```

You can use quotes inside a string, as long as they don't match the quotes surrounding the string.

more escape characters

Code	Result
\"	quote
\'	single quote
\n	New Line
\\	Backslash
\t	Horizontal Tabulator

Examples:

```
var x = 'It\'s alright.';
var x = "We are the so-called \"Vikings\" from the north.";
var x = "The character \\ is called backslash.";
var x = "This string ends with a new line \n";
```

string concatenation (+ operator)

```
var s1 = "Hello ";
var s2 = "World";
var s3 = s1 + s2;      // s3 = "Hello World";
```

- the + operator is used for string concatenation
- this can be confusing, because we often think of + as only being used for arithmetic.   But, in this case of String types, it means connect (or concatenate) two strings together.

ore string + examples

```
var s1 = "";                        // empty string
var s2 = "Abdel";
var s3 = "Zhang";

var s4 = s1 + s2;                   // result "Abdel", why? s1 is empty
var s5 = s2 + s3;                   // result "AbdelZhang"
var s6 = s2 + " " + s3;             // result "Abdel Zhang"
                               // why "Abdel" + " " + "Zhang" – adds a " "

s2 += s3;                      // s2 now equals "AbdelZhang"
                               // why? s2+=s3; is the same as s2=s2+s3;
```

string indexing [ ]

```
var str1 = "J. Trudeau";
```

Index	0	1	2	3	4	5	6	7	8	9
Character	J	.		T	r	u	d	e	a	u

Expression	Result
`str1[0]`	`"J"`
`str1[3]`	`"T"`
`str1[2]`	`" "   (space character)`
`str1.length`	`10   (be careful – why 10?)`
`str1[str1.length-1]`	`"u"`

We can think of a string as a sequence of characters.
These characters can be “indexed” starting from zero (0).

string + number types

- when the + operator is used between variables or expressions that are string and numbers, **the number type will be converted to a string.**
- Examples:

```
var str1 = "how many? ";
var strNum = "10";
var num = 10;
str1 = str1 + num;          // result is "how many? 10".
strNum = num + strNum;      // result is "1010"
strNum = "" + num;          // result is "10"
```

This **last example** is a quick trick to convert any number type into a string. "" is an empty string. Adding a number to an empty string converts the number to a string.

arrays

What is an array?

An array is a collection of variables that they all have the same name, but their indices are different.

```
var car = ["Saab", "Volvo", "BMW" ];
// We can access each individual value using the following notation.
// car[0]   is  "Saab"
// car[1]   is "Volvo"
// car[2]   is "BMW"
```

This is similar to how we can access individual characters in a String Type. Indexing starts from 0.

var len = car.length; // len is assigned 3

array can store different types

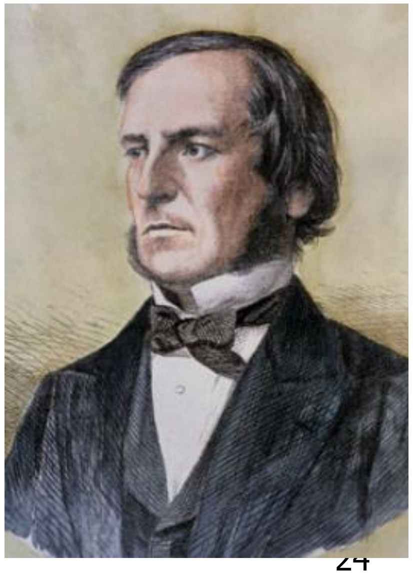
```
var car = ["Saab", "Volvo", "BMW" ]; // Array of Strings

var nums = [ 1, 2, 3, 4, 5 ];              // Array of Numbers

var data = ["EECS1012", 780, "Fall", 2018 ];  // Mix types

// data[0] is a string type with value "EECS1012"
// data[1] is a number type with value  780
```

control statements

- Program flow control is the most powerful part of programming
- It allows us to make decisions on whether to execute some statements or not
- Virtually all programming languages have some type of control statements
  - The most basic are :
    - *if statements*
    - *for or while statements (also called "loops")*

Booloean logic

- It is important to understand basic Boolean logic and expressions
- Boolean logic concerns itself with expressions that are "true" or "false"
- The name comes from the inventor, George Boole



true/false expressions

Expression	Meaning	Boolean Result
`45 < 10`	is 45 less than 10? No.	FALSE
`45 < 100`	is 45 less than 100? Yes.	TRUE
`50 > -1`	is 50 greater than -1? Yes.	TRUE
`7 == 9`	Is 7 equal to 9? No.	FALSE
`8 == 8`	Is 8 equal to 8? Yes.	TRUE

Why the crazy double ==? In JS, a single = sign means assignment. var a = 5. So, to distinguish the assignment =, from the comparison if two items are equal, we use a ==.

true/false example #2

This becomes more interesting when we use variables.

Expression	Meaning	Boolean Result

`var a = 5;`
`var b = 10;`		

`a < b`	is 5 less than 10? yes.	TRUE
`a == b`	is 5 equal to 10? no.	FALSE
`a > b`	is 5 greater than 10? no	FALSE

equality with between types

Expression	Meaning	Boolean Result

`var a = 5;`
`var b = “5”;`
	Assignment as Number
	Assignment as String	

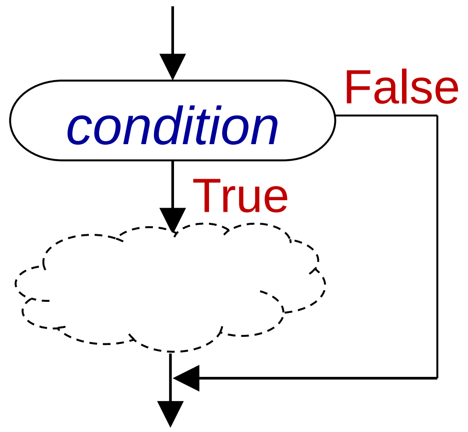
`a == b`	is 5 equal to “5”? In JS, yes!	TRUE
`a === b`	the triple-equal, tells JS to consider the type in the equality test. Is Number 5 equal to String “5” ? no.	FALSE

JS – comparison operators

Operator	Description	Comparing	Returns
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	true
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater than or equal to	x >= 8	false
<=	less than or equal to	x <= 8	true

`if` statement

```
if (condition) {
      statements;
      …
}                                    JS
```



If statements execute code within the { } if the (**condition**) expression is true.

If the expression is false, the statements within the { } are skipped.

if statement example

example

```
if (grade == "A")
{
      alert("I love EECS1012!");
}
                                                JS
```

If the variable grade is equal to “A”, then the statements are executed. Otherwise, the statements within the { … } are skipped.

`if/else` statement

```
if (condition) {
        statements1;
} else {
        statements2;
}                                                   JS
```

Almost the same as the if statement, but in this case, if the (**condition**) expression is false, *statements1* are skipped, but *statements2* are executed.

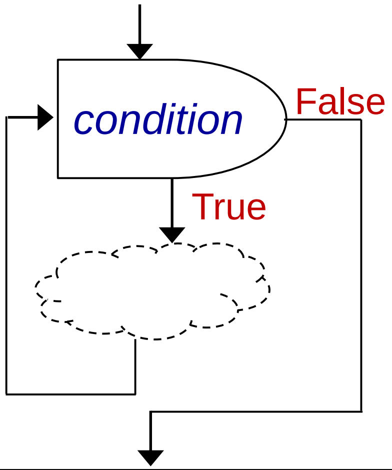
`if/else` **statement**

Example

```
if (grade == "A")
{
        alert("I love EECS1012!");
}
else
{
        alert("I hate EECS1012!");
}
                                                    JS
```

If the variable grade is equal to “A”, then the statements are executed. Otherwise the statements within the else { … } are executed.

while-loops

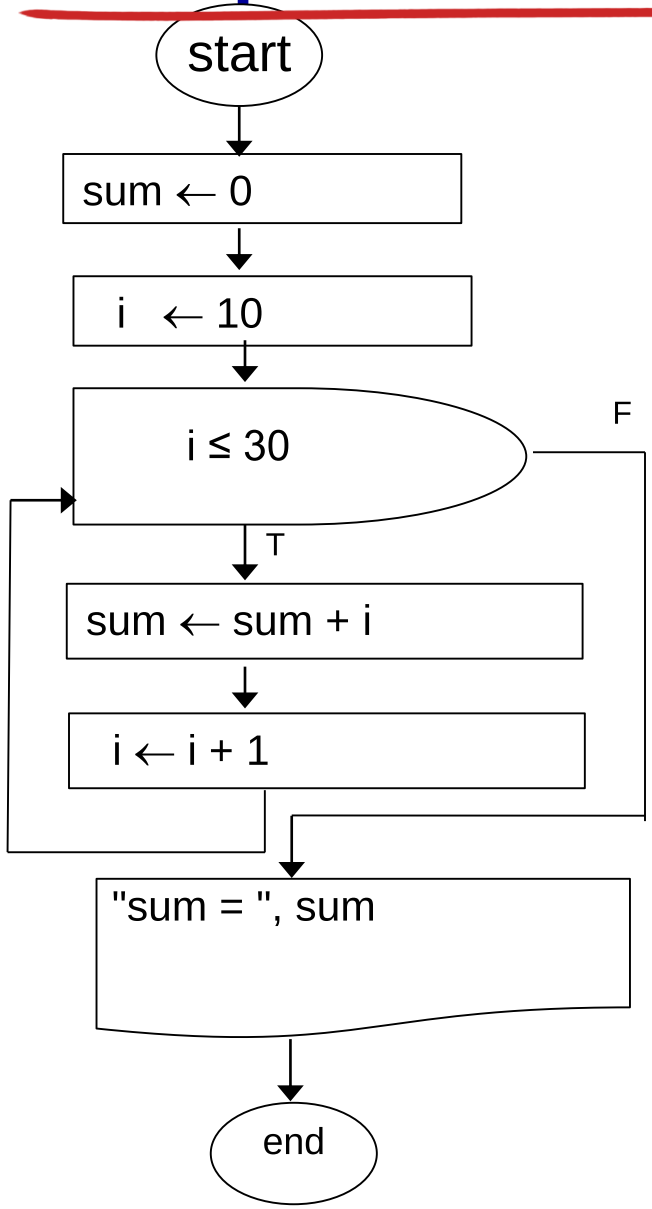


```
while (condition) {         // while the condition is true
      statements;
}                                                        JS
```

Example

```
var i=0;
var sum = 0;
while (i < 100) {   /* loop i is less than 100 is true */
      sum = sum + i;      /* adds up 0 to 99 */
      i++;                /* adds one to i   */
}                                                        JS
```

computational thinking example

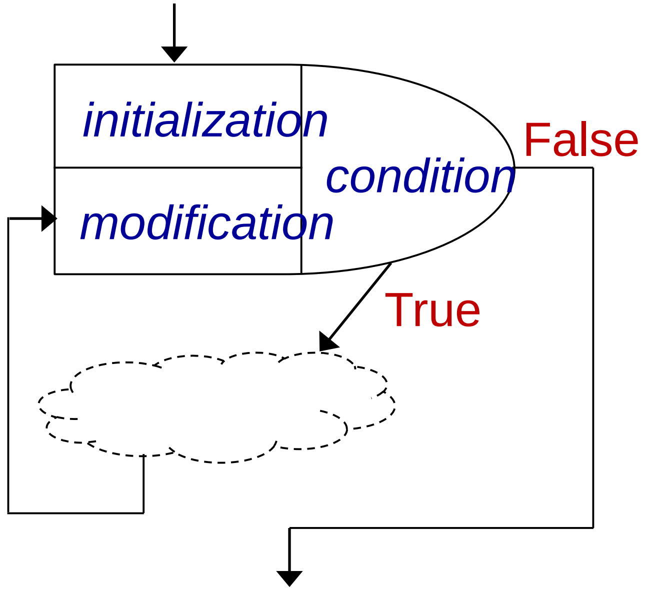


TASK
compute the sum of numbers between 10 and 30, inclusively.

```
var sum = 0;
var i=10;
while (i <= 30) {
 sum = sum + i;
 i++;
}
alert("sum ="+sum);
```

JS

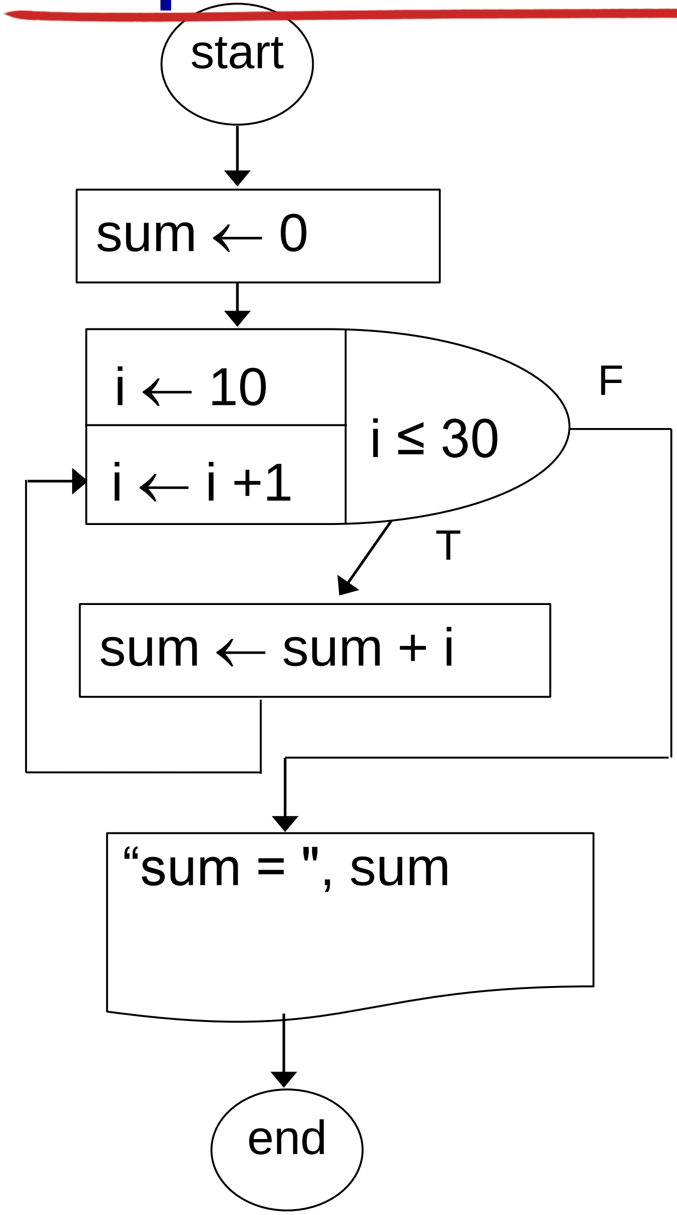
for-loop



```
for (intitialization; condition; update) {
      statements;
       ....
}                                                       JS
```

```
var s1 = "hello";
var s2 = ""; /* empty string */
for (var i = 0; i < s.length; i++) {
      s2 += s1[i] + s1[i];
}
// s2 will equal "hheelllloo"
```

computational thinking example



compute the sum of numbers between 10 and 30, inclusively.

```
var sum=0;
for(var i=10; i<= 30; i++)
{
        sum = sum + i;
}
alert("sum = "+sum);
                                  JS
```

for-loop more detail

For-loops are special cases of while-loops. They allow a fast way to specify the initialization, condition, and update rules for a while loop.

```
for (initialization; condition; update) {
        for_statements;
}
                                                        JS
```

for-loop logic as a while loop

```
initialization statement;
while (condition expression)
{

        for_statements;

        update statement;  //apply after for statements
}
```

JavaScript - functions

- We saw these before. This section gives more details
- A JavaScript function is a block of code that performs some task
- A function is executed when something "calls" or "invokes" the function

function syntax

1)
```
function name() {
 statements;
  …
}
```

Keyword function is used to define a function. name is the name of the function. The parenthesis are used to denote it is a function that accepts no parameters. See next slide.

2)
```
function name(parameter1, parameter2, …) {
 statements;
  …
}
```

This syntax allows parameters to be "passed" to the function. Parameter names are defined between the (..) (see slide 71)

3)
```
function name(parameter1, parameter2, …) {
 statements;
  …
 return value;
}
```

This syntax allows parameters to be "passed" to the function.

Note that the function **also** returns a value.

function: Ex 1

```
<html>
<head>
  <script src="example.js" type="text/javascript"></script>
</head>
<body>
<button onclick="myFunction();">Click me!</button>
</body>
</html>
```

example.js

```
function myFunction() {
       alert("You clicked my function!");
}
```

Simple function that calls an alert box with the text "You clicked my function!".

Function: Ex 2

This example combines many concepts. Functions, parameters, arrays, and if-else statements.

```
<html>
<head>
  <script src="example.js"
type="text/javascript"></script>
</head>
<body>
<button onclick="myFunction(2);">Click me!</button>
</body>
</html>
```

example.js

```
function myFunction( num ) {
   var words = ["zero", "one", "two"];
   if (num  < words.length)
   {
         alert(words[num]);
   } else
   {
        alert("more than two");
   }
}
```

The parameter name is **num**.
num is a variable that takes the value that was placed when the function was called.

If the **num** is 2 or less, then print out the word in the array. Otherwise, print out "more than two".

Function: Ex 3

```
<html>
<head>
  <script src="example.js"
type="text/javascript"></script>
</head>
<body>
<button onclick="func(2);">Click me!</button>
</body>
</html>
```

example.js

```
function doubleVar( p ) {
 var result = p + p;
 return result;
}

function func( num ) {
    var doubleValue= doubleVar( num );
    alert( "Double  " + doubleValue );
}
```

The JS file define two functions. The first, named doubleVar() takes a single parameter, named p. It computes p+p and returns it.

The second, named, myFunction(), takes a parameter, named num. The value in num is passed to the first function. The returned result is assigned to variable doubleValue. This is displayed in an alert box.

objects

- number and string variables are containers for data
- **objects** are similar, but can contain many values.
- **objects** also can associated functions (they are called methods to distinguish them from the functions you just learned about).
- We will examine several pre-defined Objects in JavaScript

Math Object

```
/*  PI is value associated with the Math object.  We access it
using the "." operator, just like we did with length for arrays
and strings.  num now equals 3.14159265358979                      */
var num1 = Math.PI;

var num2 = -50.30;
var num3 = 4;
var num4 = 66.84

var result1 = Math.round(4.7);   // method rounds a number
var result2 = Math.abs( num2 ) ; // method computes absolute value
var result3 = Math.sqrt( num3 );     // method computes the square root
var result4 = Math.min( num2, num3 ); // returns the minimum of a list of nums var
result5 = Math.max(num2, num3);  // returns the maximum of list of nums
var result6 = Math.floor(num4);      // rounds number down to nearest integer
var result7 = Math.ceil(num4);       // rounds up to nearest integer
```

More Match Object methods here: https://www.w3schools.com/js/js_math.asp

some Math methods

Function	Description
`Math.abs(n)`	absolute value
`Math.ceil(n)`	ceil means round up to the nearest integer 9.01 would round up to 10.
`Math.floor(n)`	floor means round down to the nearest integer 9.99 would round down to 9.

`Math.min(n1, n2, ..),`
`Math.max(n1,n2,..)`
	min or max of a sequence of numbers:
	e.g. max(50, 43, 1, -1, 30) = 50

`Math.sqrt(n)`	computes square root of n
`Math.random()`	return a random number between 0 (included) and 1 (excluded). So, the number will be between 0 and 0.999999999…
`Math.round(n)`	Traditional round approach, e.g. 9.4999 would round to 9; 9.50 would round up to 10.

Math object

- Random
  - Random is a Math object method that generates a returns a random floating pint number between 0 (inclusive) and 1 (exclusive)

```
//  returns a number between 0 – 1. 0 is included, but not 1.
var num1 = Math.random();

// returns a number between 0 – 5
var result = Math.floor(Math.random() * 6);

// returns a number between 0 - 100
var result = Math.floor(Math.random() * 101);
```

Date object

- Date object allows us to get information about the date.
- The format is different than the Math object. In this case, we need to use the "new" keyword to create a new Date Ojbect which is assigned to a variable.

```
var myDate = new Date();

var day  = myDate.getDay();             // returns day of the week
var year = myDate.getFullYear();        // returns the year
var month = myDate.getMonth();          // returns the month
var minute = myDate.getMinutes();       // returns the minute
var second = myDate.getSeconds();       // returns the seconds
var dateStr = myDate.toDateString();    // returns a string of the date
```

Date 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 (e.g. 2018)
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)

document object

- The document object is another useful built-in object in JavaScript. We will learn more about this in detail in upcoming lectures.
- We have use the **document** object to change the text inside a paragraph.

Example using document object

```
<!DOCTYPE html>
<html>
<head>
  <script src="example.js"
type="text/javascript"></script>
</head>
<body>
  <p id="mydata"> button not clicked </p>
  <button onclick="myFunction();" > Click Me!
</button>
</body>
</html>
```

Event ***click*** of the HTML button, calls the specified handler function

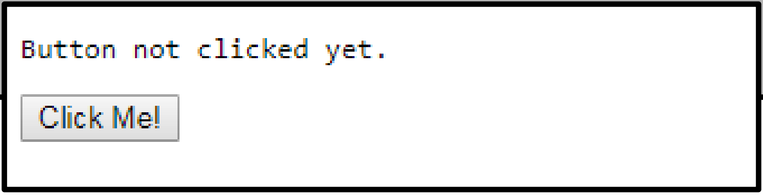
JS file: example.js

```
function myFunction() {
    var p = document.getElementById("mydata");
    p.innerHTML = "You clicked the button!";
}
```

PUTTING IT ALL TOGETHER EXAMPLES

HTML file

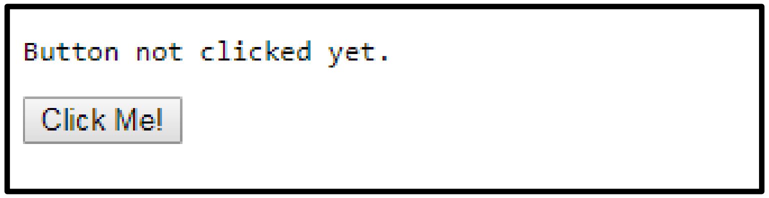
```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- link to external JS file.  Note that <script> has an
  end </script> tag -->
  <meta charset="utf-8">
  <title> Example 2 </title>
  <script src="example2.js" type="text/javascript"></script>
</head>
<body>
  <!-- Create a paragraph with id mydata -->
  <p id="mydata"> Button not clicked yet. </p>
  <button onclick="myFunction();"> Click Me! </button>
</body>
</html>
```


Button not clicked yet.

example 2 – random number

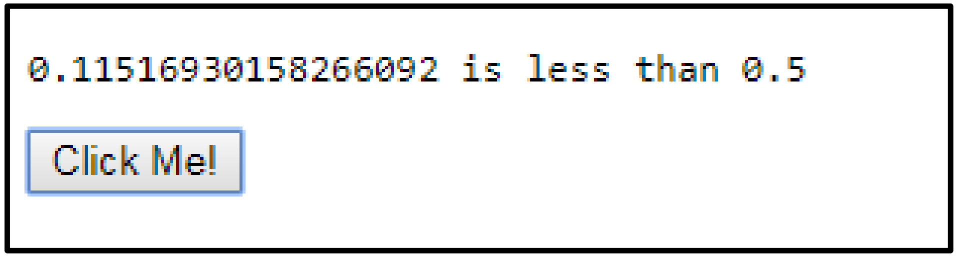
```
function myFunction()
{
  var num = Math.random();       // get a random number
  var p = document.getElementById("mydata");     // get the paragraph

  if (num < 0.5)          // if num less than 0.5
  {
        p.innerHTML = num + " is less than 0.5 ";
  }
  else
 {
        p.innerHTML = num + " is equal to or large than 0.5";
 }
}
```

example 2 – output



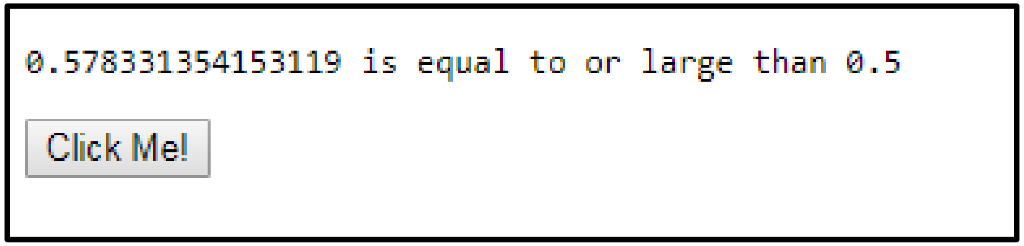
```
Button not clicked yet.
```

Before any click.



```
0.11516930158266092 is less than 0.5
```

First click calls function.
HTML of paragraph is changed.



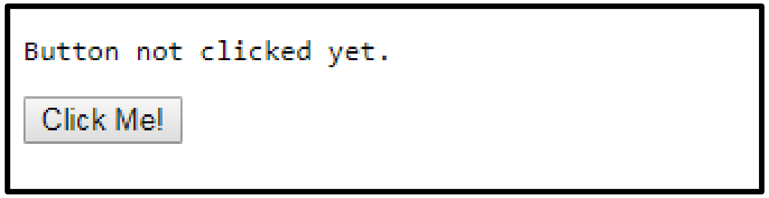
```
0.578331354153119 is equal to or large than 0.5
```

Other clicks also calls function.
HTML of paragraph is changed.

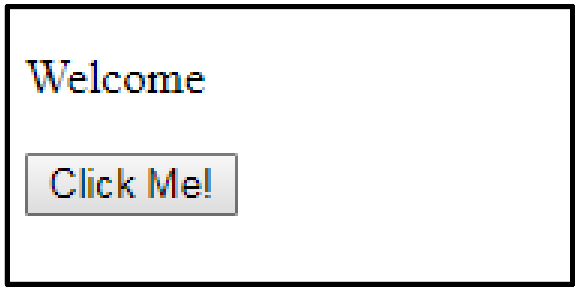
example 3 – random greeting

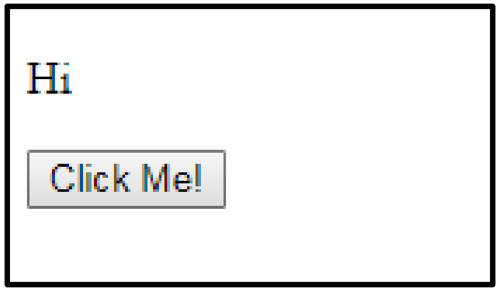
```
/* A function that returns a random number between 0 and 3 – see slide 86 */
function myRandom() {  /
        var num = Math.floor( Math.random() * 4 );
        return num;
}

/* functioned called my our HTML page when the button is clicked */
function myFunction() {
        var greetings = ["Hello", "Yo", "Hi", "Welcome"]; // declare array
        var selectOne = myRandom();     // get random number between 0 -3
        var p = document.getElementById("mydata");        // get paragraph
        p.innerHTML = greetings[ selectOne ];    // set paragraph
}
```

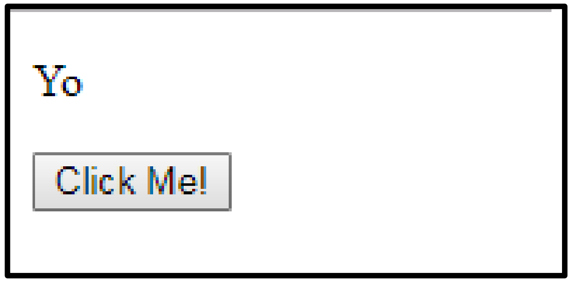
example 3 – output


```
Button not clicked yet.
```

Each click generates a new random number and otuputs the corresponding greeting in the array.


```
Welcome
```

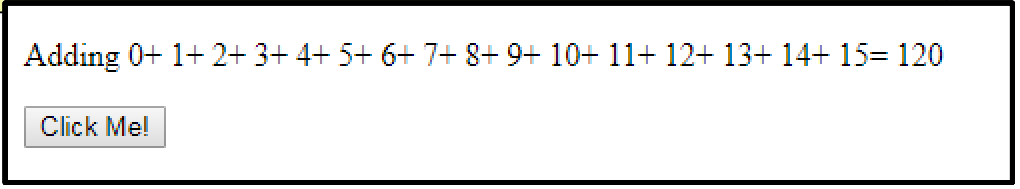

```
Hi
```


```
Yo
```

example 4 – for loops and string +

```
<p id="mydata"> Button no clicked yet. </p>
  <button onclick="myFunction(15);"> Click Me! </button>
```

```
/* called when button is clicked. Passes a value from the HTML page */
function myFunction(num)
{
        var sum = 0;
        var outputString = "Adding 0"
        var p = document.getElementById("mydata");

        for(var i=1; i <= num; i++)
        {
                sum = sum + i;
                outputString = outputString + "+" + i;
        }
        p.innerHTML = outputString + "= " + sum;
}
```



```
Adding 0+ 1+ 2+ 3+ 4+ 5+ 6+ 7+ 8+ 9+ 10+ 11+ 12+ 13+ 14+ 15= 120
```

comments on notation

In many programming languages, you will see the following notations

`value`

Text by itself is assumed to be a variable or object named *value*

`"value"`

Text with quotes is assumed to be a string with content *value*

`value()`

Text with parentheses after is assumed to be a function name *value or a method associated with an object.*