SEC201.2 Web-Based Programming

JavaScript: Introduction

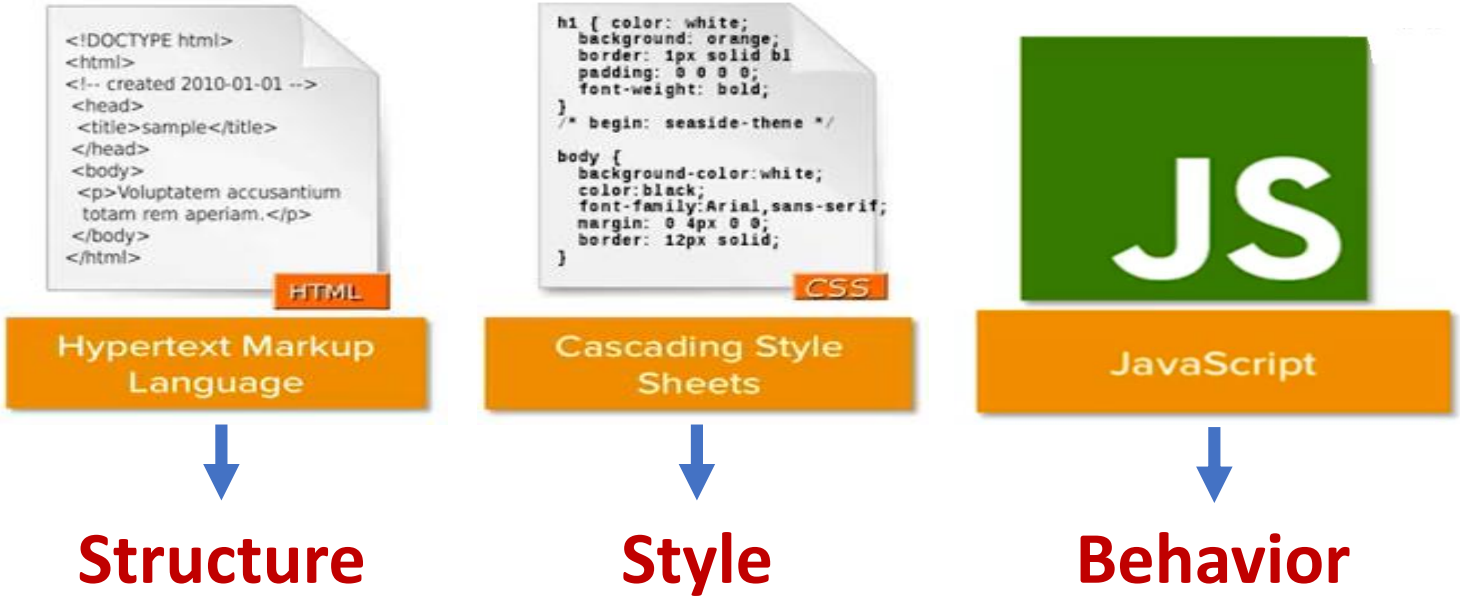
Outline

- Why Study JavaScript?
- What is JavaScript?
- Where to Place JavaScript Code?
- The JavaScript Console
- Simple Interaction
- JS Variables
- JS Data Types
- JS Events
- JS Functions
- Handling Bugs
- Making Decisions (IF, SWITCH)
- Loops
- Global vs. Local Variables
- Logical Operators
- Arrays
- Generating Random Numbers
- An Example JS Project

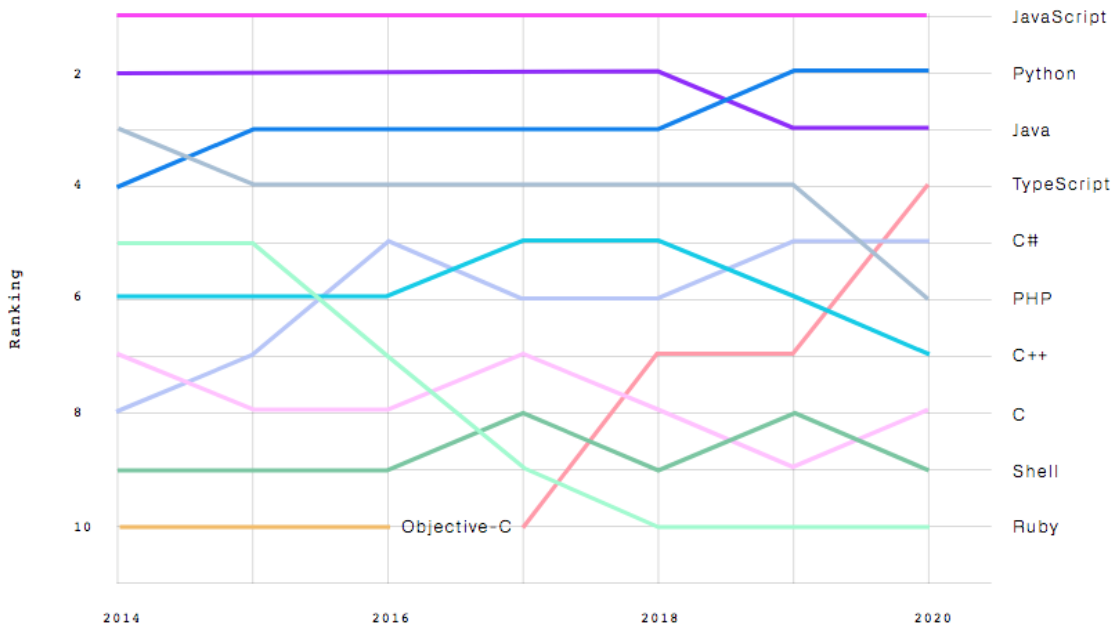
Why Study JavaScript?

JavaScript is one of the **3 languages** all web developers **must** learn:

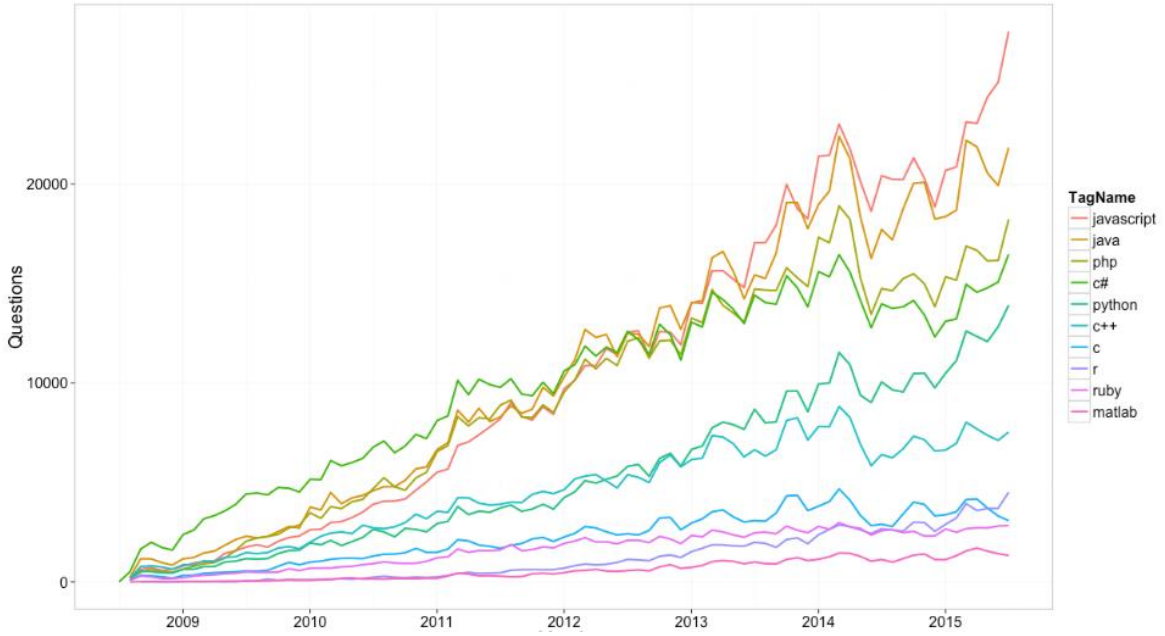
1. **HTML** to define the **structure** of web pages
2. **CSS** to specify the **style/layout** of web pages
3. **JavaScript** to program the **behavior** of web pages



JavaScript Popularity



Top Languages over the years
Source: [The State of the Octoverse | GitHub]



Number of questions in Stack Overflow
Source: [http://blog.revolutionanalytics.com/2015/07/the-most-popular-programming-languages-on- stackoverflow.html]

JavaScript is the dominant web programming language

What is JavaScript?

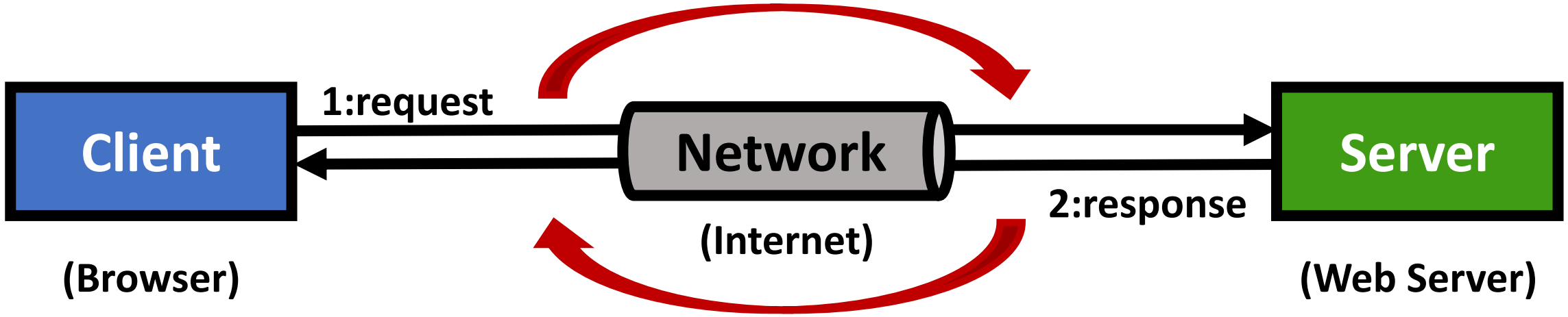
- JavaScript is an **interpreted programming language**, <u>not a compiled language</u>
  - A program such as C++ or Java needs to be compiled before it is run
    - The source code is passed through a program called a **compiler**, which translates it into **bytecode** that the machine understands and can execute
  - In contrast, **JavaScript** <u>has no compilation step</u>
    - Instead, an interpreter in the browser reads over the JavaScript code, interprets each line, and runs it
    - More modern browsers use a technology known as Just-In-Time (JIT) compilation, which compiles JavaScript to executable bytecode just as it is about to run

- Java and JavaScript are two **entirely distinct languages**
  - The most significant difference between them
    - Java is a compiled language, and JavaScript is an interpreted language
    - JavaScript runs on many browsers out-of-the-box, whereas Java applets require an additional plug in
    - Both languages have different runtime environments, different governing bodies, different libraries

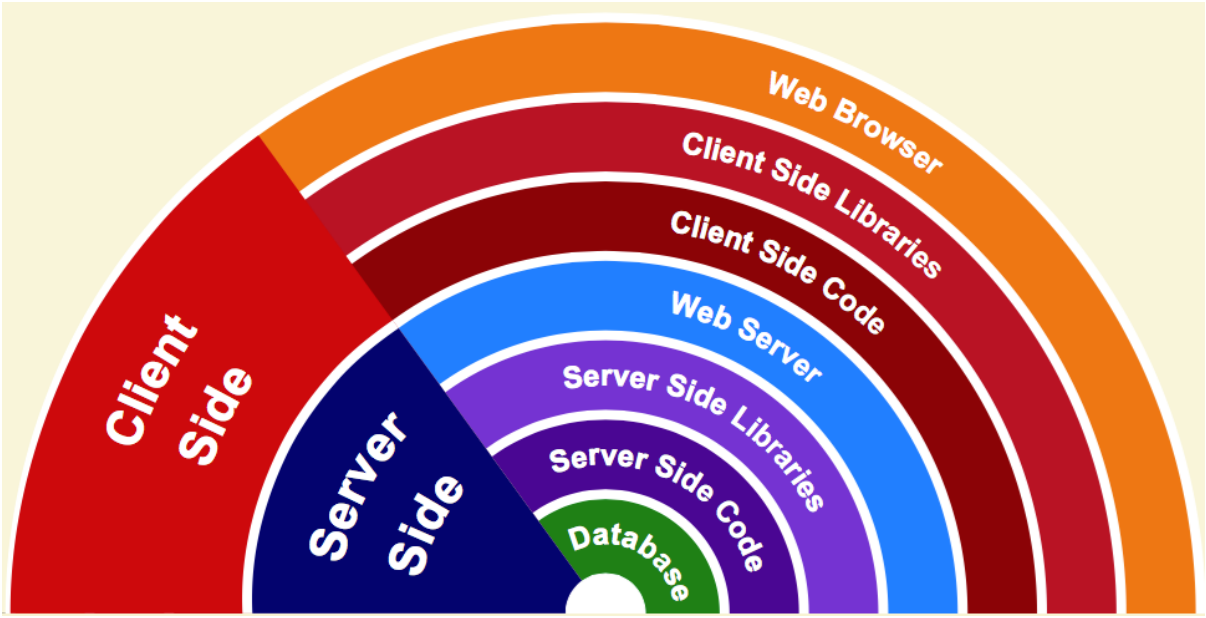
What is JavaScript?

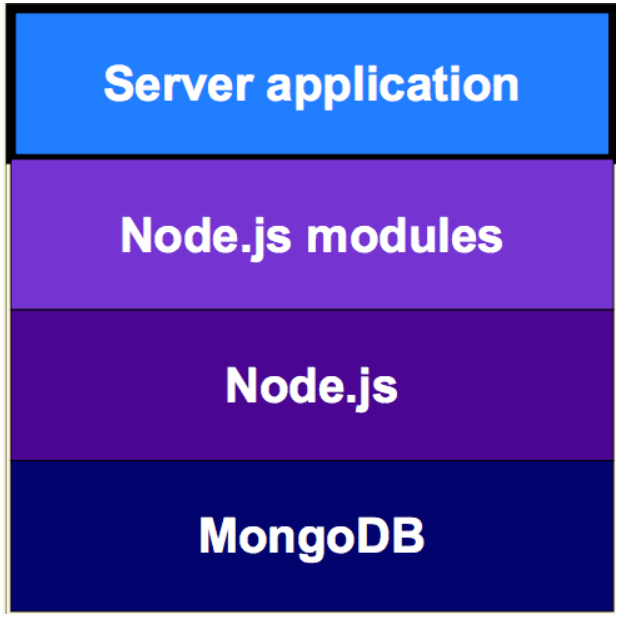
- JavaScript is primarily a **client-side language**
  - A webpage can contain embedded JavaScript, which executes when a user visits the page
  - The language was created to allow web developers to embed executable code on their webpages, so that they could make their webpages interactive, or perform simple tasks
  - Today, browser scripting remains the main use-case of JavaScript

- JavaScript is a very **useful and common** programming language
  - JavaScript runs in every web browser, out of the box
  - JavaScript applications run on every device, whereas desktop or mobile applications run only on the application it is targeted to (Windows, Mac OSX, Linux, iPhone, Android)
  - So, JavaScript allows us to write cross-platform apps in a really easy way
  - JavaScript's role has also expanded significantly over time
    - Platforms such as "Node.js" allow developers to run JavaScript server-side
    - It is now possible to create entire web applications in which both client-side and server-side logic is written in JavaScript



Request – Response Cycle





Where to Place JavaScript Code?

- In HTML, JavaScript code must be inserted between **<script>** and **</script>** tags
- Where to place these **<script></script>** tags?
  - JavaScript code can go almost anywhere
    - JavaScript in the **<head>** section
    - JavaScript in the **<body>** section

  OR

    - External JavaScript (***without the <script> tags***)

External JavaScript

- External scripts are practical when the same code is used in many different web pages
- JavaScript files have the file extension **.js**
- To use an external script, put the name of the script file in the **src** (source) attribute of a **<script>** tag

```
<script src="myScript.js"></script>
```

- You can place an external script reference in **<head>** or **<body>** as you like
- You can add several script files to one page by using several script tags

```
<script src="myScript1.js"></script>
<script src="myScript2.js"></script>
```

- External scripts can be referenced with a full URL or with a path relative to the current web page

External JavaScript Advantages

Placing scripts in external files has some advantages:

- It separates HTML and code
- It makes HTML and JavaScript easier to read and maintain
- Cached JavaScript files can speed up page loads

The JavaScript Console

- The nice thing about interpreted languages is that they are designed to be run with a single pass through the source code, running each instruction step-by-step
  - Means that we can give the interpreter a single step and ask it to run it

- There are several JavaScript consoles that allow us to do this
  - Most browsers have one available
  - Think of it as a command-line interface that runs JavaScript on your JavaScript engine

Chrome, **Safari** and **Opera**
  - Open a new tab. Right click on the page and click **Inspect Element**. Click on **Console**

Firefox
  - Open the **Tools** menu. Go to **Web Developer > Web Console**

- Behind the console is the ***read-eval-print loop*** (**REPL**)
  - Refers to the loop that the console runs: it first reads your input, then it evaluates it as JavaScript code, then it prints the results immediately

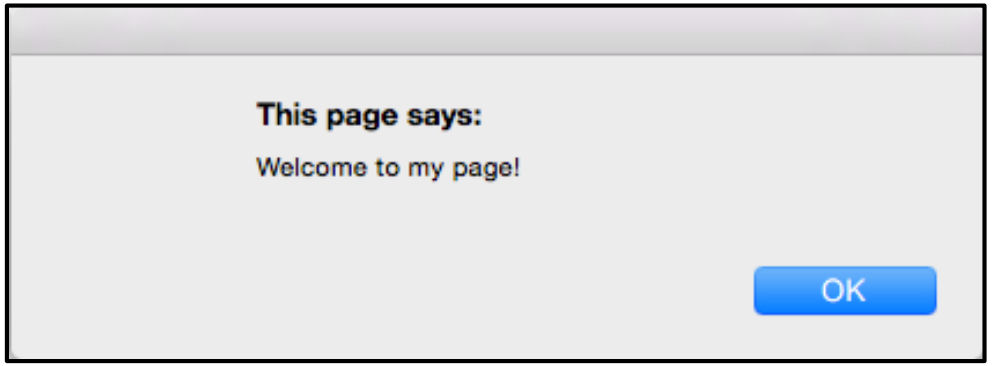
Simple Interaction

- There are 3 JavaScript **popups**
  - **alert()**
  - **confirm()**
  - **prompt()**

Alert() – Show a Message

- **alert()** shows text to the user

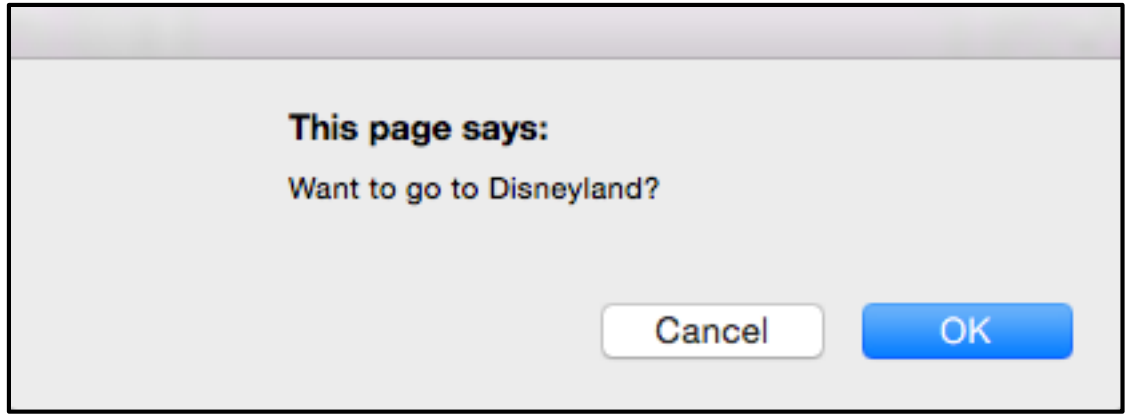
```html
<!doctype html>
<html>
    <head>
        <title>Example of alert()</title>
        <script>
            alert("Welcome to my page!");
        </script>
    </head>
</html>
```



Confirm() – Making a Decision

- **confirm()** displays a popup box with a message, along with an **OK** and a **Cancel** button

```
<!doctype html>
<html>
    <head>
        <title>Example of confirm()</title>
        <script>
            if (confirm("Want to go to Disneyland?"))
                document.location.href
                ="http://park.hongkongdisneyland.com";
        </script>
    </head>
</html>
```



Prompt() – Simple Text Input

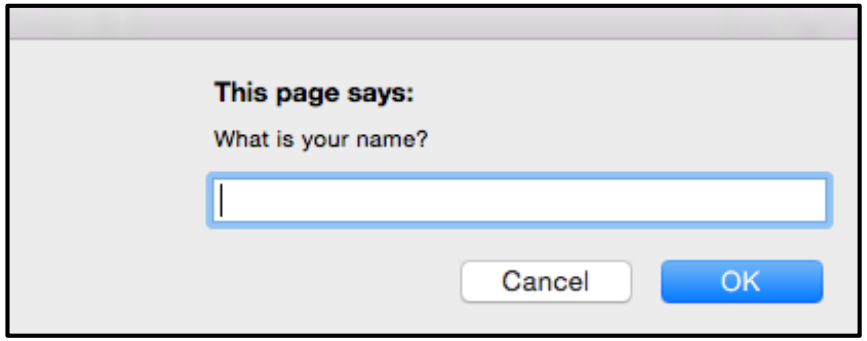
- For getting input from the user, you can use **prompt()**

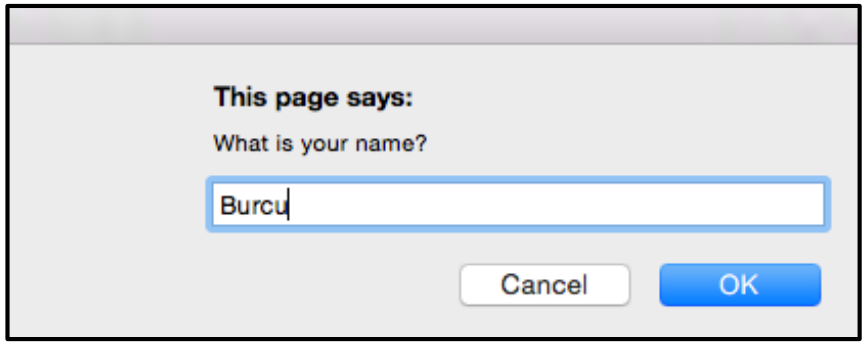
```
<script>
    var user_name; //Create a variable
    user_name = prompt("What is your name?");
</script>
```

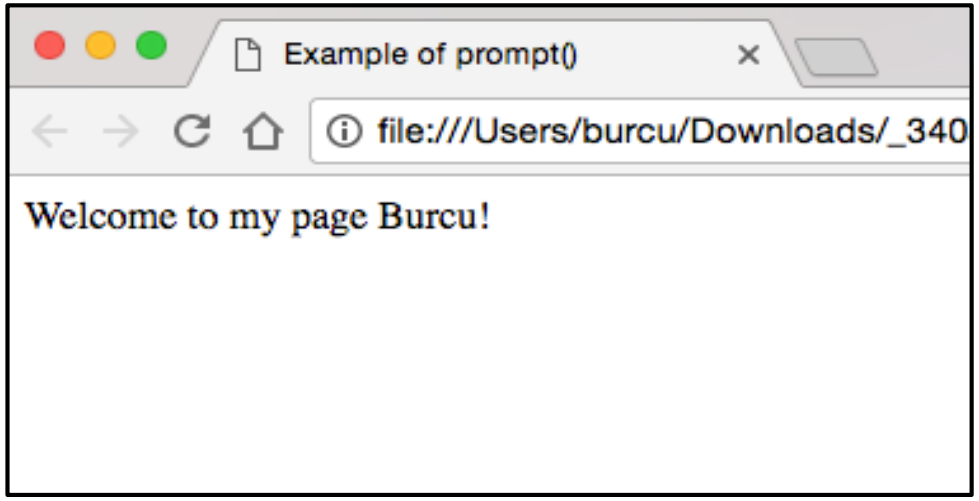
- You don't have to create a variable before you use it
- However, it is good habit to get into

Example: Prompt()

```
<!doctype html>
<html>
    <head>
        <title>Example of prompt()</title>
        <script>
            var user_name; //Create a variable

            user_name = prompt("What is your name?");
            document.write("Welcome to my page "
                            + user_name + "!" );
        </script>
    </head>
</html>
```







JavaScript Variables

- A variable is like a box or container for storing data values
- You can make a variable and put something in it

```
var x = 5;
var y = 6;
var z = x + y;
```

- Later, you can take it out of the box and use it
- You can change what is stored in the box any time

JavaScript variables are containers for storing data values!

JavaScript Identifiers

- All JavaScript **variables** must be **identified** with **unique names**
  - These unique names are called **identifiers**
- Identifiers can be short names (like x, y) or more descriptive names (age, sum, totalVolume)
- The general rules for constructing names for variables (unique identifiers)
  - Names can contain letters, digits, underscores, and dollar signs
  - Names must begin with a letter
  - Names can also begin with $ and _
  - Names are case sensitive (y and Y are different variables)
  - Reserved words (like JavaScript keywords) cannot be used as names

JavaScript identifiers are case-sensitive!

JavaScript Data Types

- JavaScript variables can hold many **data types**
  - **Number**
  - **String**
  - **Boolean**
  - **Object (3 types of objects: Object, Date, Array)**
  - **Function**
  - **Null & undefined (cannot contain values)**

Number

- JavaScript has only one type of number
- Can be written with or without a decimal place

```
var number1 = 34.289; // Written with decimals
var number2 = 100;    // Written without decimals
```

- Can use scientific notation

```
var big_number = 123e5;       // 12300000
var small_number = 123e-5;    // 0.00123
```

String

- A string simply means text
- You can use **single** or **double** **quotes**

```
var name = "Mert";          // Using double quotes
var title= 'Lecturer';      // Using single quotes
```

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

```
var answer = "It's alright";    // Single quote inside double
var answer = "He is 'Johnny'"; // Single quotes inside double
var answer = 'He is "Johnny"'; // Double quotes inside single
```

Boolean

- A Boolean value can only be **true** or **false**

```
var condition1 = true;
var condition2 = false;
```

- Booleans are often used in conditional testing
- Do not confuse Boolean values with String values

```
var myBool = true;              // Boolean type
var myString = "true";          // String type
```

Objects

- JavaScript objects are written with **curly braces**
- Object properties are written as **name:value** pairs, separated by **commas**

```
var person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue”
};
```

- The object (person) in the example above has 4 properties
  - firstName, lastName, age, and eyeColor

A Variable Type can Change

- JavaScript has **dynamic types** → same variable can be used to hold different data types

  Ex: If you do this

```
var storage = ”Dave"; // Now storage is a String
```

  And then this

```
storage = 98;          // Now storage is a Number
```

- The type of the variable is **immediately changed**

More on JavaScript Types & Variables

- If you re-declare a JavaScript variable, it will not lose its value

Ex:
```
var carName = "Volvo";
var carName;
```

→ The variable carName will still have the value "Volvo" after the execution of these statements!!

- If you put a number in quotes, the rest of the numbers will be treated as strings, and concatenated

Ex:
```
var x = "5" + 2 + 3;     // 523
var x = 2 + 3 + "5";    // 55
```

- When adding a number and a string, JavaScript will treat the number as a string

Ex:
```
var x = 16 + 4 + "Volvo";  // 20Volvo
var x = "Volvo" + 16 + 4;  // Volvo164
```

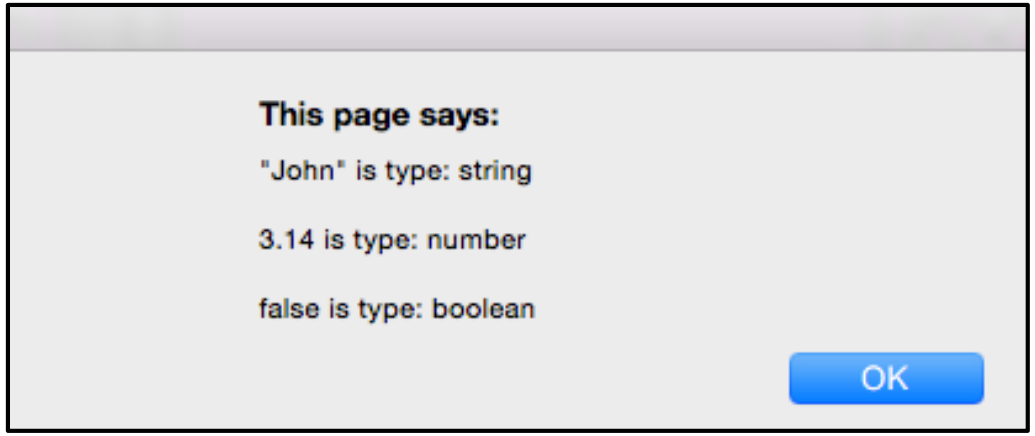
The Typeof Operator

- You can use the **typeof** operator to check the type of a variable
- The **typeof** operator returns the type of a variable or an expression

```
typeof "John”        // Returns "string”
typeof ""            // Returns "string"
typeof 0             // Returns "number”
typeof 3.14          // Returns "number"
typeof (3 + 4)       // Returns "number"
```

Example: Typeof

```
<!doctype html>
<html>
    <head>
        <title>Variable Type Example</title>
    </head>
    <body>
        <script>
            alert( '"John" is type: ' + typeof "John" + "\n\n"
                   + "3.14 is type: " + typeof 3.14 + "\n\n"
                   + "false is type: " + typeof false ) ;
        </script>
    </body>
</html>
```



Common Changes

Code	Quicker Typing
`count = count + 1`	`count++`
`count = count - 1`	`count--`
`count = count + 10`	`count += 10`
`hello = hello + "!"`	`hello += "!"`
`marks = marks - 20`	`marks -= 20`
`pigs = pigs * 5`	`pigs *= 5`
`cakes = cakes / students`	`cakes /= students`

From One Type to Another

Function	Meaning
`parseInt()`	Converts to an integer
`parseFloat()`	Converts to a floating point number
`String()`	Converts the value of an object to a string

Events

- An event is when something happens – **“things”** that happen to HTML elements
- For example:
  - Click on something
  - Move the mouse
  - Press a key on the keyboard
- You can arrange for some code that you write to be executed when the event occurs
- Some common HTML events
  - **onload** → The browser has finished loading the page
  - **onclick** → The user clicks an HTML element
  - **onmouseover** → The user moves the mouse over an HTML element
  - **onkeydown** → The user pushes a keyboard key

Onload Event

- **onload** is triggered when the object has loaded

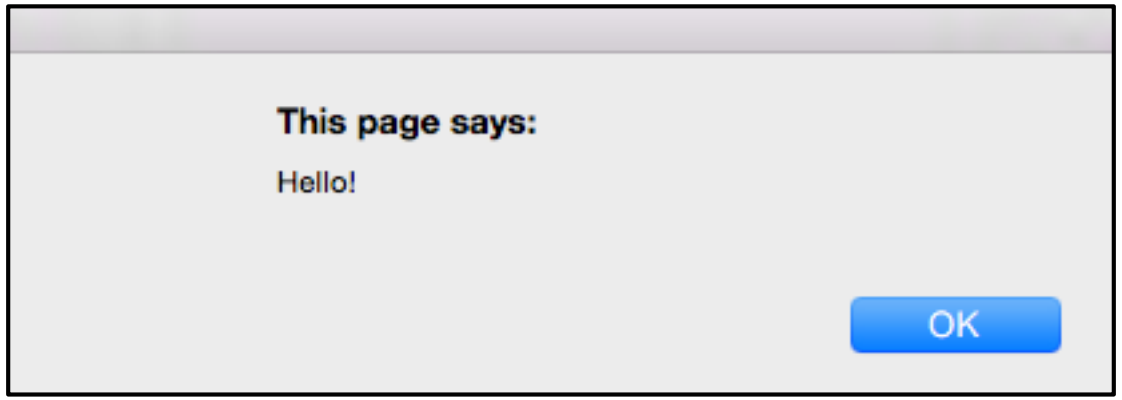
```
<body onload="alert('Hello!')">
```

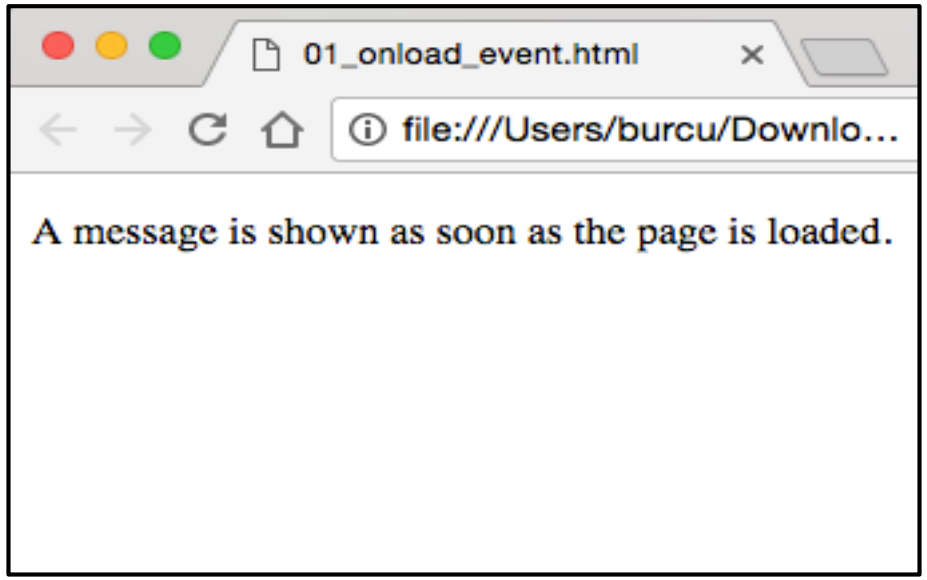
… the main web page content goes here …

```
</body>
```

Example: Onload Event

```html
<!doctype html>
<html>
    <body onload="alert('Hello!')">
        <p>
            A message is shown as soon
            as the page is loaded.
        </p>
    </body>
</html>
```



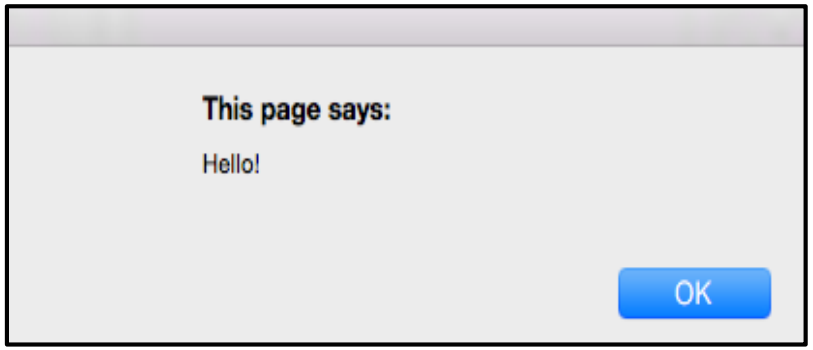


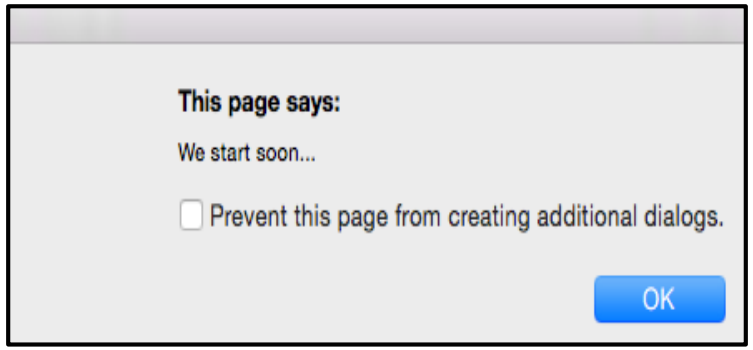
Example: Onload Event – 3 popup windows

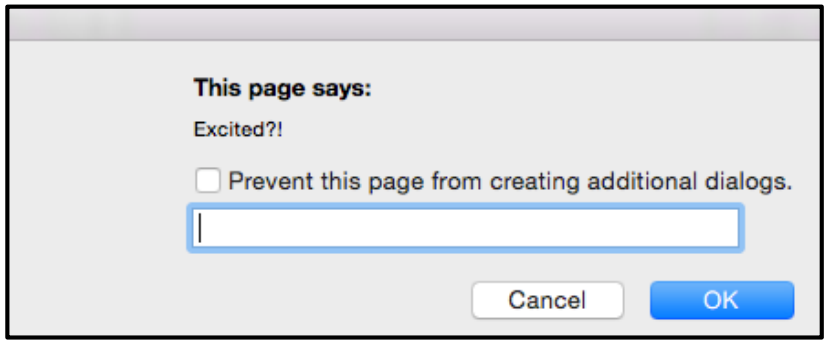
- You can execute as much code as you like

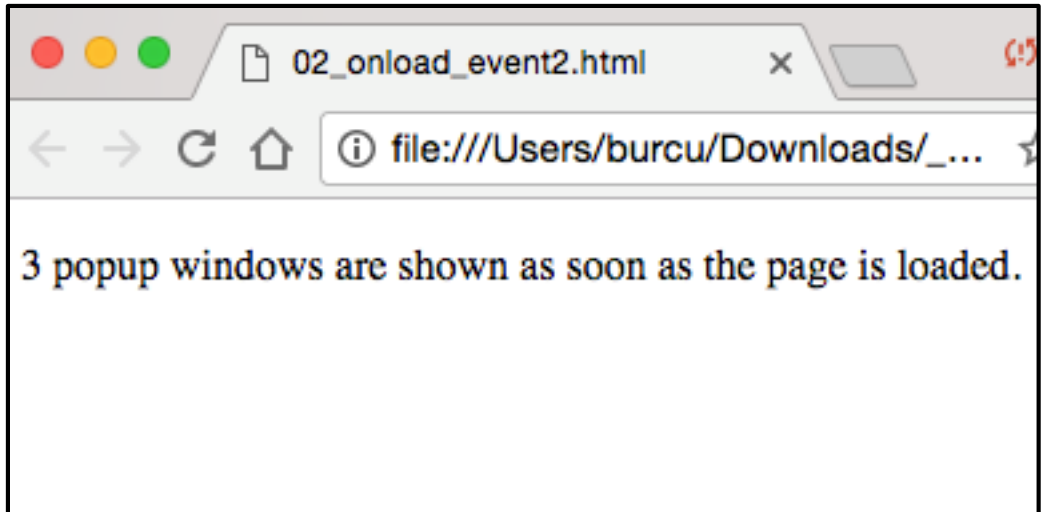
```
<!doctype html>
<html>
    <body onload="alert('Hello!');
                  alert('We start soon...');
                  prompt('Excited?!') ">
        <p>
            3 popup windows are shown as
            soon as the page is loaded.
        </p>
    </body>
</html>
```

Example: Onload Event – 3 popup windows









JavaScript code is often several lines long. It is more common to see event attributes calling ***functions***

Functions

- A function is a group of code:

```
function do_something() {
```

… code goes here …

```
}
```

- Run the function like this:

```
do_something();
```

Example: Function

```
<!doctype html>
<html>
    <head>
        <title>Example of a function</title>
        <script>
            function greet_the_user(){
                alert('Hello!');
                alert('We start soon...');
                prompt('Excited?!')
            }
        </script>
    </head>
    <body onload="greet_the_user()">
    </body>
</html>
```

Function Parameters

- You can pass something to a function

```
function purchase (cats) {
```

… code here uses cats …

```
}
```

- Run the function like this:

```
purchase(10);
```

Function Response

- You can get a response from a function

```
function do_something () {
```

… code here stores something in answer …

```
    return answer;
}
```

- Use the function like this:

```
result = do_something();
```

Example: Function Response

```
<!doctype html>
<html>
    <body onload="check_user_age()">
        <h1>This is my naughty home page.</h1>
        <script>
            function check_user_age(){
                if (age_of_user() < 18)
                    alert("Please go to another page.");
            }
            function age_of_user(){
                var age_text, age;
                age_text=prompt("What is your age?");
                age=parseInt(age_text);
                return age;
            }
        </script>
    </body>
</html>
```

Recursive Functions

- A function can call itself

```
function do_something( control_value ) {
```

… code here calls ***do_something(…)***

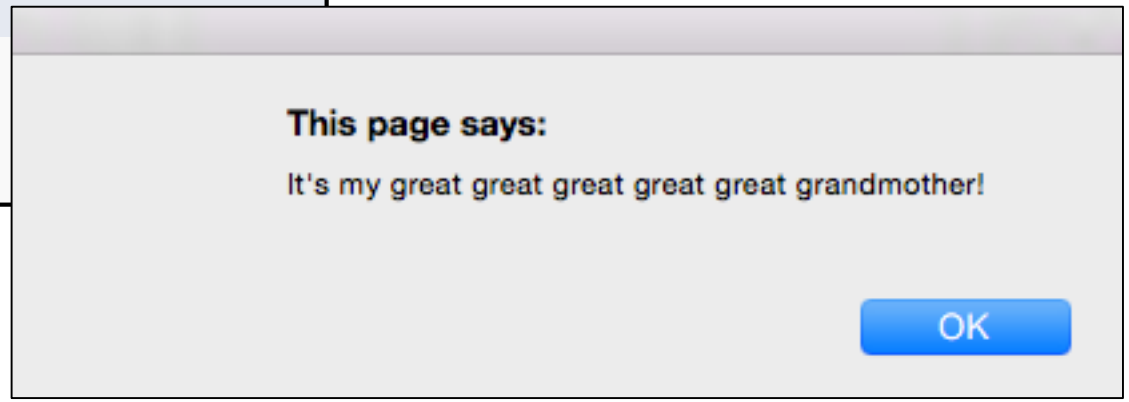
```
}
```

- Start the function like this:

```
result = do_something( 10 );
```

Example: Recursive Function

```
<!doctype html>
<html>
    <body>
        <script>
            alert("It's my " + build_great(5) +
                  "grandmother!");

            function build_great( depth ) {
                if (depth > 0)
                    return "great " + build_great( depth - 1 );
                else
                    return "";
            }
        </script>
    </body>
</html>
```



Handling Bugs

```html
<!doctype html>
<html>
    <head>
        <title>Handling Bugs</title>

        <script>

            //Comments in JavaScript

            /*
            You can inspect your JavaScript code with your browser

            Chrome, Safari and Opera
                Open a new tab. Right click on the page and click Inspect Element. Click on Console
            Firefox
                Open the Tools menu. Go to Web Developer > Web Console
            */

            var user_name "";
            user_name = promptttt("What is your name?");
            alert("The value entered was: " + user_name);
            document.write("Welcome to my page " + user_name + "!");

            // IMPORTANT: One of the interesting things is that you can try out any JavaScript you like in the Console.

            /* Console is very useful. In the above example we use alert to help us understand what's happening. Basically, we
            used it for debugging. However, that's not very good, because anybody who goes to this web page will see that
            alert window. It'll be better if only you or other developers could see the message.

            We can do that using console, console.log --> replace alert with console.log
                        console.log("The value entered was: " + user_name);
            And now, we see exactly the same message right in the Console. But people who are regular users of the browser
            page can't see it.
            */


        </script>
    </head>

    <body>
    </body>
</html>
```

Making Decisions

You can make decisions using

- **if** statements
- **switch** statements

COMPARING THINGS	
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

```
if                          switch … case
if … else                   default
if … else if … else
if … else if … else if … else
```

Example: IF

```
<!doctype html>
<html>
    <head>
        <script>
            var user_name;

            user_name=prompt("What is your name?");
            if (user_name == "bob")
                alert("Great name!");
        </script>
    </head>
</html>
```

- You must use **braces { }** for more than 1 line of code

```
if (user_name == "bob") {
    alert("Great name!");
    awesome_name = true;
}
```

- Braces are <u>optional</u> if there is **only one line** of code

Example: IF … ELSE IF … ELSE

```
<!doctype html>
<html>
    <head>
        <script>
            var user_name;

            user_name=prompt("What is your name?");
            if (user_name == "bob")
                alert("Great name!");
            else if (user_name == "dave")
                alert("Pretty good name!");
            else if (user_name == "oz")
                alert("Excellent name!");
            else
                alert("Your name isn't great, never mind...");
        </script>
    </head>
</html>
```

SWITCH

- Used for a series of comparison

```
switch(variable_name) {
    case "option_1": do_something_1();
        break;

            ...    :    ...

    case "option_n": do_something_n();
        break;

    default: do_something_default();
}
```

- **break** is used to stop any more case comparisons

Example: Switch

```
<!doctype html>
<html>
    <head>
        <script>
        var user_name = prompt("What country would you like to visit?");
        switch(user_name) {
            case "Canada":
            case "France":
                alert("Take me also!");
                break;
            case "Japan":
            case "Philippines":
                alert("Great! Have fun!");
                break;
            case "North Korea":
                alert("Oh! Good luck!");
                break;
            default:
                alert("I am sure you will have a great time");
        }
        </script>
    </head>
</html>
```

While Loops

- A loop repeats some code again and again

while
do … while

- A **while loop** is the simplest loop

```
while (condition) {

    // ... code goes here ...

};
```

- Each time the loop content is executed we call it an **iteration**

INDEXOF()

→ `string.indexOf("text")`

Gives the location of the first “text” in the string

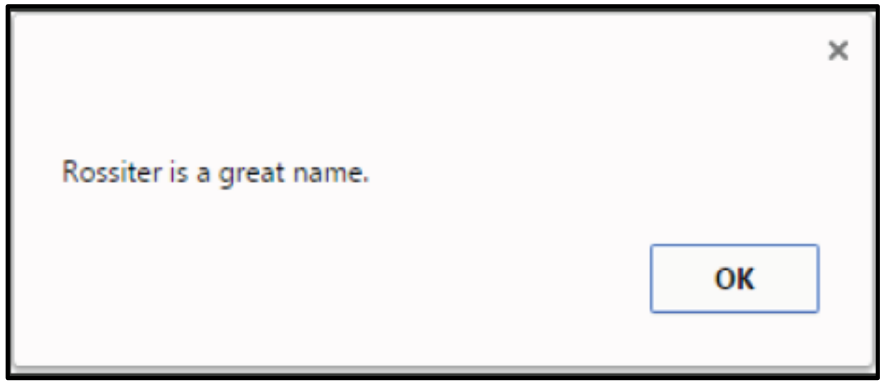
<u>Ex:</u>

```
var text = "The cat's hat was wet";
result = text.indexOf("at");
```

=> result will be 5

Example: While Loop & IndexOf()

```
<!doctype html>
<html>
    <head>
        <title>Example of while()</title>
        <script>
            var response, finished;
            finished=false;
            alert("Rossiter is a great name.");
            while (!finished){
                response=prompt("Do you agree?");
                if (response.indexOf("y")==0)
                    finished=true;
            }
        </script>
    </head>
</html>
```



do … while() Loops

- do … while is an “upside-down” version of while

```
do {

    // ... code goes here ...

} while(condition);
```

- A **do … while** loop will execute <u>at least once</u>

Example: do … while() Loop

```
<!doctype html>
<html>
    <head>
        <title>Example of do .. while()</title>
        <script>
            var response, finished;
            finished=false;
            alert("Rossiter is a great name.");
            do {
                response=prompt("Do you agree?");
                if (response.indexOf("y")==0)
                    finished=true;
            } while (!finished);
        </script>
    </head>
</html>
```

More On Variables: Global vs. Local Variables

Local Variables

- Variables declared within a function can only be accessed within the function
- They are *local* to the function, and so are called local variables

```
<!doctype html>
<html>
    <body>
        <script>
            function show_money() {
                var money = 2;

                alert("In the function, the value is: "+ money);
            }
            money = 99;

            alert("In the main part, the value is: "+ money);
            show_money();
            alert("In the main part, the value is: "+ money);
        </script>
    </body>
</html>
```

More On Variables: Global vs. Local Variables

Global Variables

- The opposite of a local variable is a ***global*** variable
- Global variables are created in the main part
- They can work inside or outside functions

```
<!doctype html>
<html>
    <body>
        <script>
            function show_money() {
                alert("In the function, the value is: "+ money);
            }
            var money = 99;

            alert("In the main part, the value is: "+ money);
            show_money();
            alert("In the main part, the value is: "+ money);
        </script>
    </body>
</html>
```

More On Variables: Global vs. Local Variables

Local and Global Variables Sharing the Same Name

- JavaScript will give priority to the local variable inside the function

Creating Global Variables Inside Functions

- If you assign a value to a variable that has not been declared, it will automatically become a global variable

```
<!doctype html>
<html>
    <body>
        <script>
            function show_money() {
              money = 2;

              alert("In the function, the value is: "+ money);
            }
            show_money();
            alert("In the main part, the value is: "+ money);
        </script>
    </body>
</html>
```

Logical Operators

Boolean

- A ***Boolean*** value is either **true** or **false**
- A variable which has a Boolean value is called a **Boolean variable**

Logical Operators

- Logical operators work with Boolean values
- JavaScript has these logical operators
  - Logical **AND** – the **&&** operator
  - Logical **OR** – the **||** operator
  - Logical **NOT** – the **!** operator

AND – &&

a	b	a && b
false	false	false
false	true	false
true	false	false
true	true	true

- && - the result is **true** if <u>both inputs are true</u>, <u>otherwise</u>, the result is **false**

- Short-Circuit in AND
  - JavaScript is clever
  - When it evaluates an AND it checks the first input
  - If the value is false, it knows the result must be false
  - So, it doesn't bother checking the next input

```html
<!doctype html>
<html>
    <body>
        <script>
            var you_are_rich = false;
            var you_have_partner = true;
            var you_have_flat = true;
            var life_is_fantastic = you_are_rich
                                    && you_have_partner
                                    && you_have_flat;

            alert("life is fantastic is " +
                    life_is_fantastic);
            you_are_rich = true;
            life_is_fantastic = you_are_rich
                                && you_have_partner
                                && you_have_flat;
            alert("life is fantastic is now " +
                    life_is_fantastic);
        </script>
    </body>
</html>
```

OR – ||

a	b	a || b
false	false	false
false	true	true
true	false	true
true	true	true

- || - the result is **false** if <u>both inputs are false</u>, <u>otherwise,</u> the result is **true**

- Short-Circuit in OR
  - If JavaScript is evaluating OR and the first input is true, it knows the result must be true
  - So, it doesn't bother checking the second input

```
<!doctype html>
<html>
    <body>
        <script>
            var you_are_rich = false;
            var you_have_partner = true;
            var you_have_flat = false;
            var life_is_good = you_are_rich
                               || you_have_partner
                               || you_have_flat;

            alert("life is good is " + life_is_good);
            you_have_partner = false;
            life_is_good = you_are_rich
                           || you_have_partner
                           || you_have_flat;
            alert("life is good is now " + life_is_good);
        </script>
    </body>
</html>
```

NOT – !

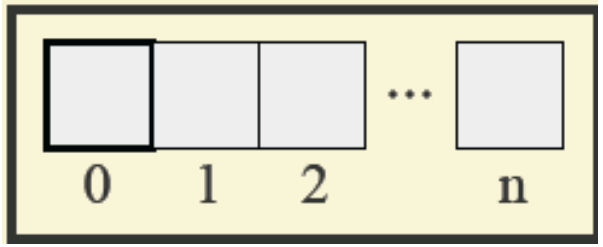
- ! - the result is the **opposite** of the input

a	!a
false	true
true	false

```
<!doctype html>
<html>
    <head>
        <title>Not Operator Example</title>
    </head>
    <body>
        <script>
            var you_are_male = true;
            var you_are_female = !you_are_male;

            alert("you_are_male is " + you_are_male);
            alert("you_are_female is " + you_are_female);
        </script>
    </body>
</html>
```

Arrays

- An array is a linear continuous storage



- You can think array as a group of boxes
- Each box has a unique identity, which is called an **index**
- The index of the first box is **0**

How to Create an Array?

- Here is how you create a new array with 3 boxes:

```
var pets = ["Dog", "Cat", "Rabbit"];
```

- You can create a new array with 10 boxes without any element inside the boxes like this:

```
var pets = new Array(10);
```

- You can put anything in an array
- Any element can be any data type

JOIN()

- Use ***array*.join(*separator*)** to <u>convert *array* into string</u>

```
var pets = ["Dog", "Cat", "Rabbit"];
alert(pets.join(" and "));
// This shows "Dog and Cat and Rabbit"
```

- ***separator*** is by default ","

```
var pets = ["Dog", "Cat", "Rabbit"];
alert(pets.join());
// This shows "Dog,Cat,Rabbit"
```

Retrieving Element From An Array

- Let's assume that we have the following array

```
var pets = ["Dog", "Cat", "Rabbit"];
```

- We can retrieve elements like this:

```
alert(pets[2]); // This shows "Rabbit"
```

Changing An Element of An Array

- Let’s assume that we have the following array

```
var pets = ["Dog", "Cat", "Rabbit"];
```

- We can change something stored in the array like this:

```
pets[2] = "Hamster";
// Now pets is ["Dog", "Cat", "Hamster"]
```

More On Arrays

Array Size

- We can get the size of an array (i.e., how many boxes it has) using ***array.length***

```
var pets = ["Dog", "Cat", "Rabbit"];
alert(pets.length); // This shows 3
```

Adding to the End

- Add a new element to the **end** of an array with ***array.push()***
- The index is automatically updated

```
var pets = ["Dog", "Cat", "Rabbit"];
pets.push("Hamster");
// Now pets is
// ["Dog", "Cat", "Rabbit", "Hamster"]
```

More On Arrays

Adding to the Front

- Add a new element to the **front** with ***array.unshift()***
- The index is automatically updated

```
var pets = ["Dog", "Cat", "Rabbit"];
pets.unshift("Hamster");
// Now pets is
// ["Hamster", "Dog", "Cat", "Rabbit"]
```

Removing from the Back

- To remove an element from the **end**, use ***array.pop()***
- pop() returns the removed element, so result is “Rabbit”

```
var pets = ["Dog", "Cat", "Rabbit"];
var result = pets.pop();
// Now pets is ["Dog", "Cat"]
```

More On Arrays

Removing From the Front

- ***array.shift()*** removes an element from the **front**
- shift() returns the removed element, so result is “Dog”
- The index is automatically updated

```
var pets = ["Dog", "Cat", "Rabbit"];
var result = pets.shift();
// Now pets is ["Cat", "Rabbit"]
```

Combining Two Arrays

- Use ***array1.concat(array2)*** to combine two arrays into one

```
var pets = ["Dog", "Cat", "Rabbit", "Hamster"];
var primes = [2, 3, 5, 7, 11];
var result = pets.concat(primes);
// result is ["Dog", "Cat", "Rabbit", "Hamster", 2, 3, 5, 7, 11]
```

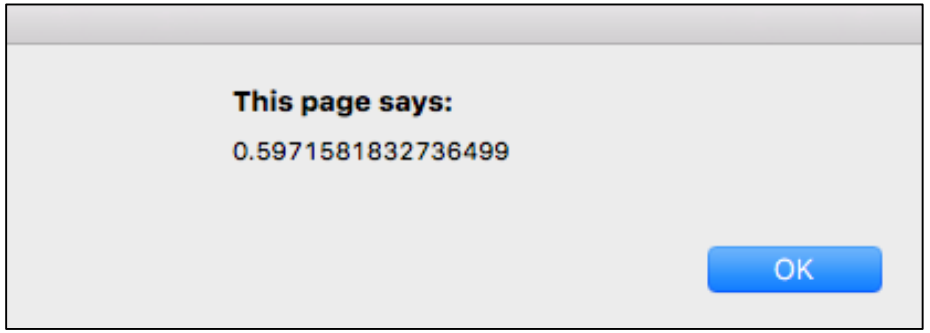
Generating Random Numbers

- You can generate a random number like this:

```
var random_number = Math.random();
```

- The resulting range is [0,1)
- 1 will not be generated

```
<!doctype html>
<html>
    <body>
        <script>
            var random_number;

            random_number = Math.random();
            alert( random_number );
        </script>
    </body>
</html>
```



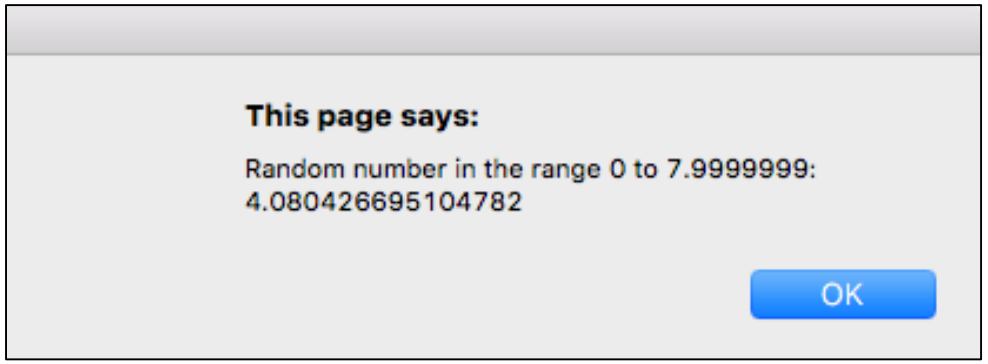
Setting up the Range

- So far, the random number is in the range 0 up to 1
- Multiply in order to get the range you want, i.e.

```
random_number = Math.random() * max_value;
```

- We now have a number in the range [0, max_value)

```
<!doctype html>
<html>
    <body>
        <script>
            var random_number;

            random_number = Math.random() * 8;
            alert("Random number in the range 0 to " +
                  "7.9999999:\n" + random_number );
        </script>
    </body>
</html>
```

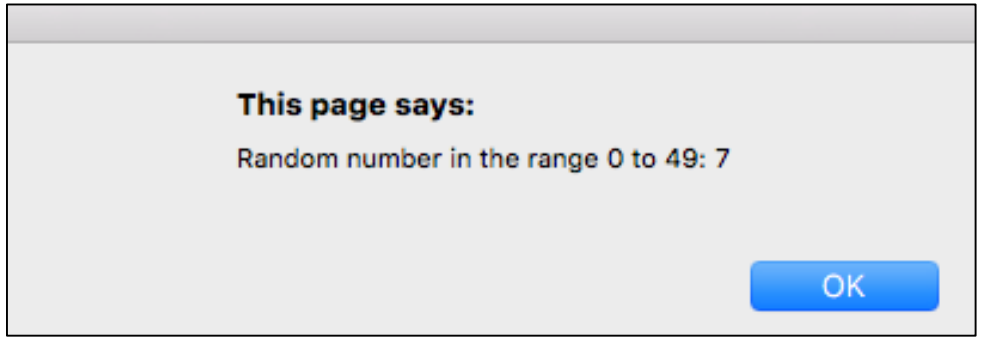


Throw Away the Decimal Place

- There is still a decimal place
- ***Math.floor()*** dumps the decimal place

  Ex: 2.82248 becomes 2

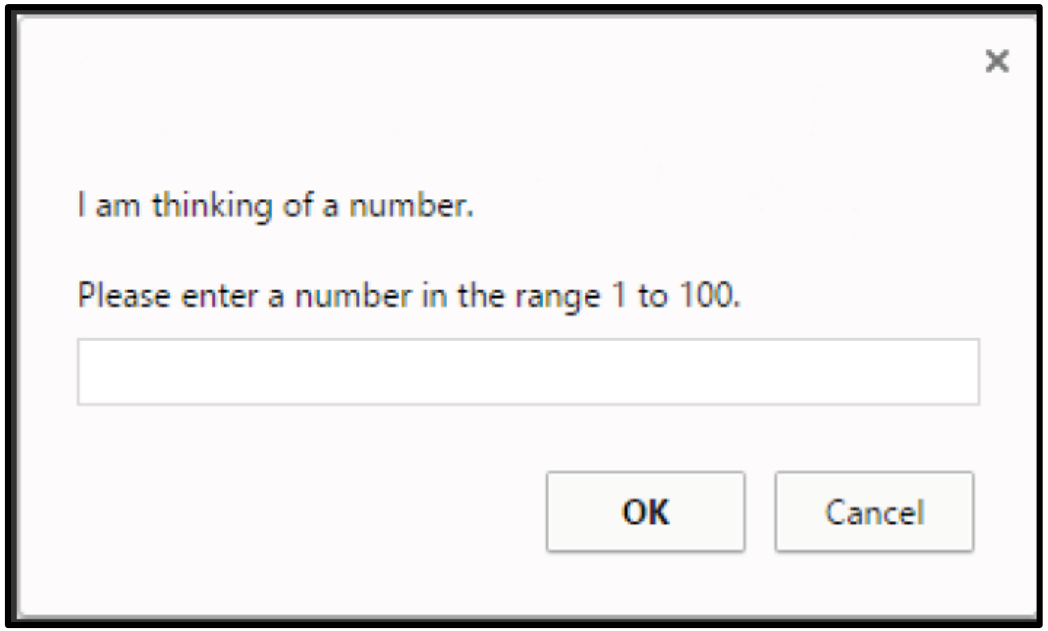
```html
<!doctype html>
<html>
    <body>
        <script>
            var random_number;

            random_number = Math.random() * 50;
            random_number = Math.floor( random_number );
            alert("Random number in the range 0 to 49: " +
                    random_number);
        </script>
    </body>
</html>
```



An Example JavaScript Project: Guessing Game

How it Works

- The computer thinks of a number in the range [1, 100]
- The player must guess what it is
- The computer tells the player if answer is right or wrong
- When the game is over, the player is told how many times they guessed



An Example JavaScript Project: Guessing Game

Flow Chart