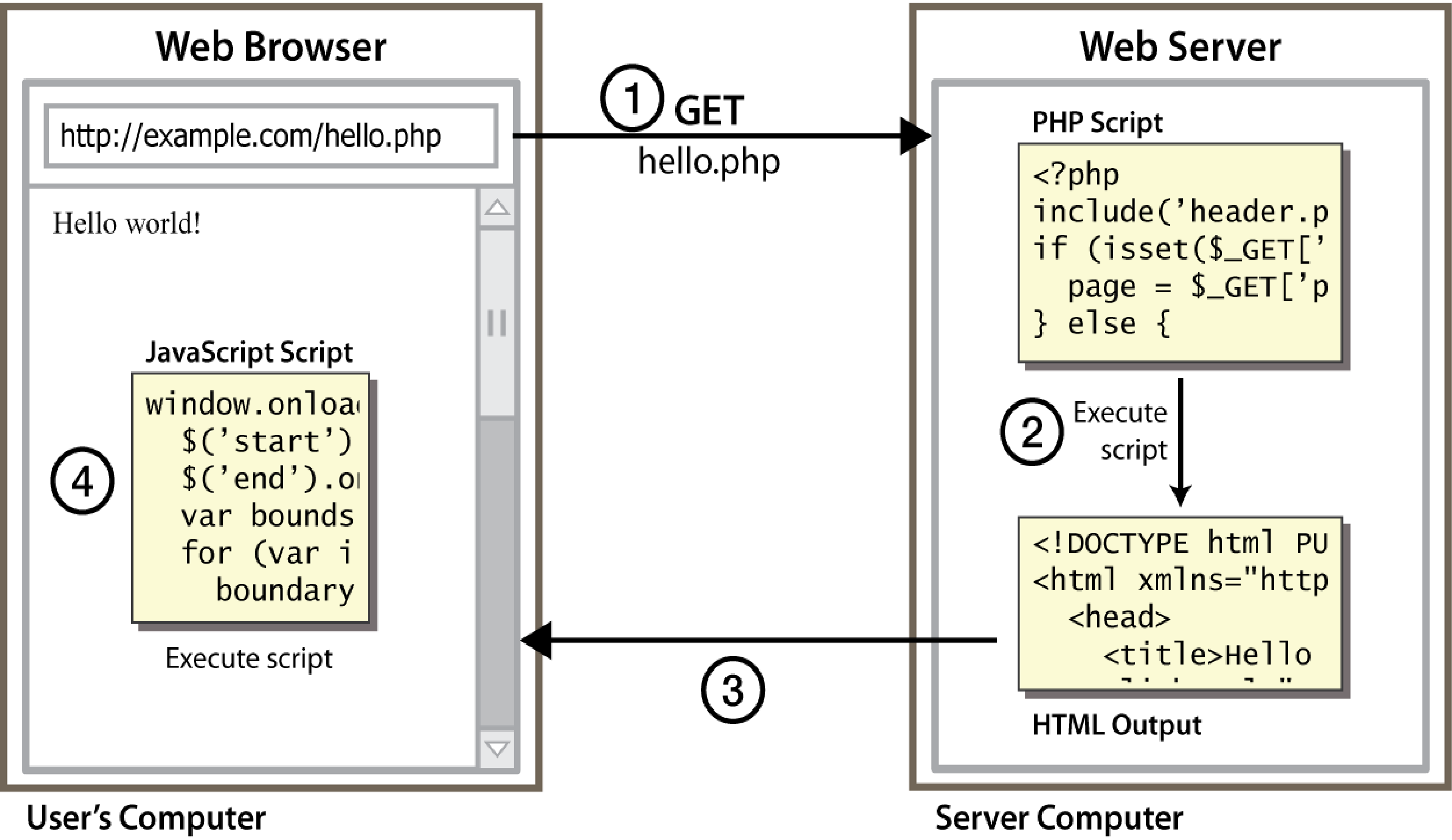
JavaScript

Client side scripting



```
window.onloa
  $('start')
  $('end').o
  var bounds
  for (var i
    boundary
```

```
<?php
include('header.p
if (isset($_GET['
  page = $_GET['p
} else {
```

```
<!DOCTYPE html PU
<html xmlns="http
  <head>
    <title>Hello
```

Why we use Client side scripting?

Server side programming already allows us to create dynamic web pages. Why also use client-side scripting?

Client-side scripting (JavaScript) benefits:

- **usability**: can modify a page without having to post back to the server (faster UI)
- **efficiency**: can make small, quick changes to page without waiting for server
- **event-driven**: can respond to user actions like clicks and key presses

Server-side programming (PHP, Java, Microsoft dotNet, nodeJs) benefits:

- **security**: has access to server's private data; client can't see source code
- **compatibility**: not subject to browser compatibility issues
- **power**: can write files, open connections to servers, connect to databases, ...

What is JavaScript

A lightweight programming language ("scripting language")

- used to make web pages interactive
- insert dynamic text into HTML (ex: user name)
- react to events (ex: page load user click)
- get information about a user's computer (ex: browser type)
- perform calculations on user's computer (ex: form validation)
- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

History

- Created by Brendan Eich in 1995 for Netscape Navigator 2 release. Called Mocha and later LiveScript.
- On release of 1.1 name was changed to JavaScript.
- In 1997 JavaScript 1.1 was submitted to ECMA as a proposal. TC39 was assigned to standardize the syntax and semantics of the language.
- TC39 released ECMA-262 known as ECMAScript.
- ECMAScript is basis for all the JavaScript implementations.

History

- ECMAScript is the official name of the language.
- ECMAScript versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6.
- Since 2016 new versions are named by year (ECMAScript 2016 / 2017 / 2018)

THE THREE LAYERS OF THE WEB

- HTML for Content
  - <p class=“warning”>There is no <em>download link</em> on this page.</p>

- CSS for Presentation
  - .warning { color: red; }

- JavaScript for Behavior
  - <script type=“text/javascript”> window.alert(document.getElementsByClassName(“ warning”)[0].innerHTML);

    </script>

LIBRARIES

JavaScript can be used to create reusable libraries. Some are:

- Prototype
- Script.aculo.us
- Yahoo! User Interface Library
- Dojo
- jQuery
- MooTools
- Knockout

PROGRAMMING WITH JAVASCRIPT

Running a JavaScript program/script. There are two ways:

- <script type="text/javascript">
  window.alert("JavaScript");
  </script>
- <script type="text/javascript"
  src="myScript.js"></script>

There was once a
language attribute

also in the script tag.
i.e.,
language="javascript
1.5" or whatever

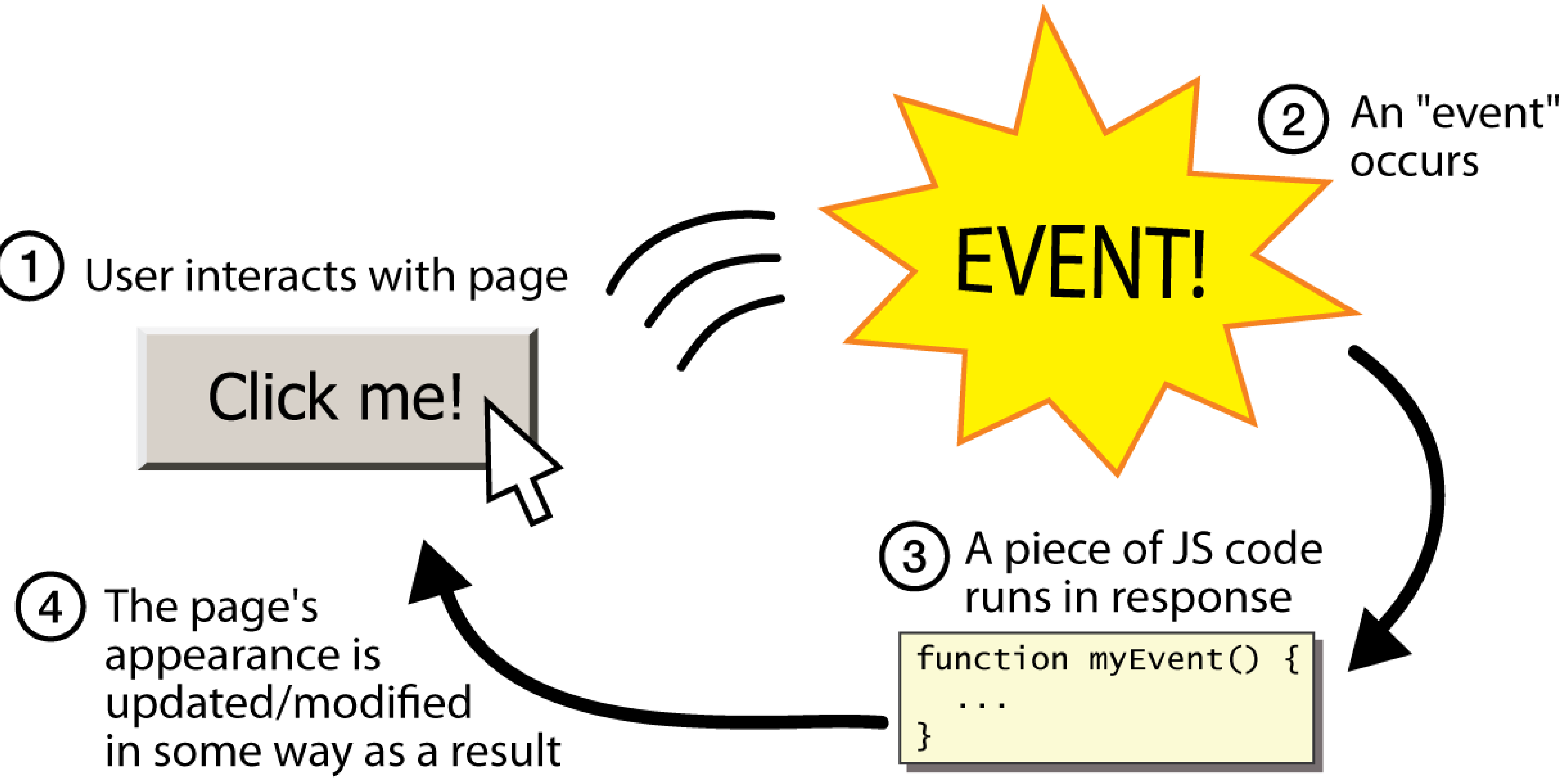
- <script type="text/javascript" src="myScript.js"
  defer></script>

Linking to a JavaScript file: `script`

```html
<script src="filename" type="text/javascript"></script>
```
HTML

- script tag should be placed in HTML page's head
- script code is stored in a separate .js file
- JS code can be placed directly in the HTML file's body or head (like CSS)
  - but this is bad style (should separate content, presentation, and behavior

Event-driven programming


1 User interacts with page
Click me!
EVENT!
2 An "event" occurs
3 A piece of JS code runs in response
4 The page's appearance is updated/modified in some way as a result

```
function myEvent() {
  ...
}
```

Event-driven programming

- you are used to programs start with a main method (or implicit main like in PHP)
- JavaScript programs instead wait for user actions called *events* and respond to them
- event-driven programming: writing programs driven by user events
- Let's write a page with a clickable button that pops up a "Hello, World" window...

Buttons

```html
<button>Click me!</button>                          HTML
```

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse 1. click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control

STATEMENTS, COMMENTS AND VARIABLES

- Statements: Can be separated by new line or semicolon.
- Comments: 2 types.
  - Single line: // Author: Manvendra SK
  - Multi line: /* Perhaps some lengthy
    license. */
- Variables: Overwrites when redefined. 2 types.
  - Local: Declared using var keyword.
  - Global: Declared without var keyword or attached to **window** object directly. i.e., window.someVar = “value”;
  - We use **assignment opertor (=)** to assign values.

VARIABLE DATA TYPES

JavaScript is **loosely typed** language. While Java is **strictly typed**.

- Numbers: 3, -3, 3.33, -3.33. These all are numbers in JS, whether it's some **integer** or **floating point number**.
  - Operations: **addition (+), subtraction (-), division (/), multiplication (*), modulo division (%)**
  - Result is always promoted to float whenever possible. i.e., 5 / 3, 3 / 5, 0.5 * 5, 0.5 + 5
  - Assignment operators: **+=, -=, /=, *=, %=**
  - Special operators: **increment (++), decrement (–)**

STRINGS

- Strings: Series/sequence of characters from zero to infinity. Can contain any character. Can be created using single or double quotes.
- Use backslash (\) to escape special characters.
  i.e. “Hi this is **\“**special**\”** word here.”
  “Hi this is **\t**tab**\t** here.“
- Operations: Concatenation.
  i.e., “Some ” + “string.”
  “Some ” + someVar + “.”
  var name = “Manvendra”;
  name = name + “SK”; or name += “SK”;

`String` type

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" ")); // "Connie"
var len = s.length; // 13
var s2 = 'Melvin Merchant';
                                                          JS
```

- methods: charAt, charCodeAt, fromCharCode, indexOf, lastIndexOf, replace, split, substring, toLowerCase, toUpperCase
  - charAt returns a one-letter String (there is no char type)
- length property (not a method as in Java)
- Strings can be specified with "" or ''
- concatenation with + :
  - 1 + 1 is 2, but "1" + 1 is "11"

More about `String`

- escape sequences behave as in Java: \' \" \& \n \t \\
- converting between numbers and Strings:

```
var count = 10;
var s1 = "" + count; // "10"
var s2 = count + " bananas, ah ah ah!"; // "10 bananas, ah
ah ah!"
var n1 = parseInt("42 is the answer"); // 42
var n2 = parseFloat("booyah"); // NaN                    JS
```

- accessing the letters of a String:

```
var firstLetter = s[0]; // fails in IE
var firstLetter = s.charAt(0); // does work in IE
var lastLetter = s.charAt(s.length - 1);                 JS
```

Splitting strings: split and join

```
var s = "the quick brown fox";
var a = s.split(" "); // ["the", "quick", "brown", "fox"]
a.reverse(); // ["fox", "brown", "quick", "the"]
s = a.join("!"); // "fox!brown!quick!the"
                                                        JS
```

- split breaks apart a string into an array using a delimiter
  - can also be used with regular expressions (seen later)
- join merges an array into a single string, placing a delimiter between them

BOOLEANS

- Booleans: Are you saying **true** or **false**.
  - Cases for true
    - "<1 space here>", "string", "undefined", 1, 2, 3, -3, -2, -1, true; all represents **true** value.
  - Cases for false
    - "<empty string>", undefined, null, void(0), 0, false; all represents **false** value.
  - Universal checking program:
    - <any value> ? "True" : "False";

Special values: null and undefined

```js
var ned = null;
var benson = 9;
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```

JS

- `undefined` : has not been declared, does not exist
- `null` : exists, but was specifically assigned an empty or null value
- Why does JavaScript have both of these?

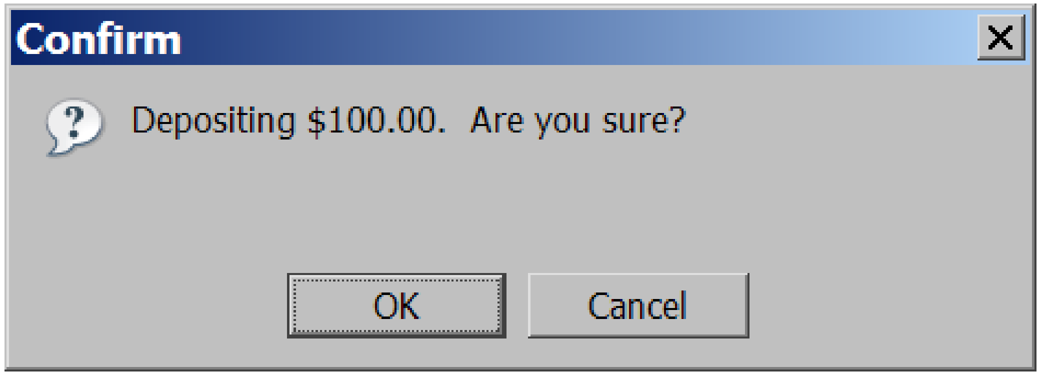
Math object

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
                                                        JS
```

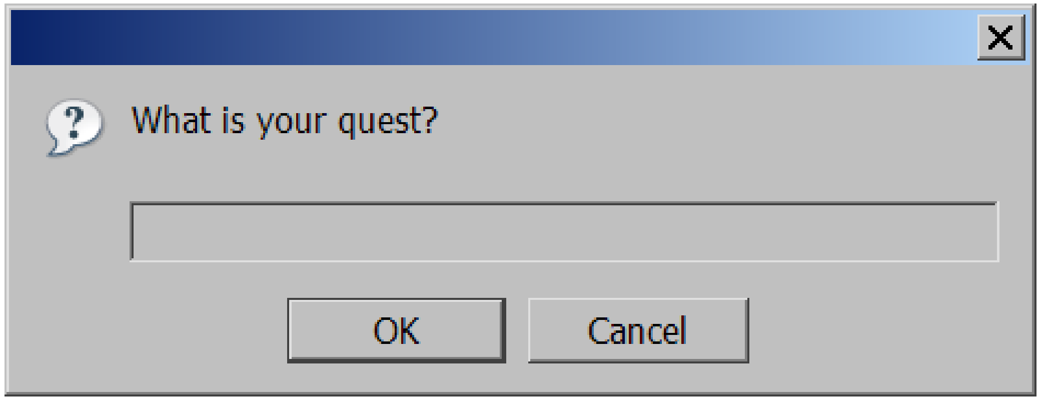
- **methods:** `abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan`
- **properties:** `E, PI`

Popup boxes

```
alert("message"); // message
confirm("message"); // returns true or false
prompt("message"); // returns user input string
```
JS


Confirm

Depositing $100.00. Are you sure?


What is your quest?

ARRAYS

- Arrays: To hold a collection of items together. Can store any data types together in single array.

  i.e., var array = [1, {“k”: “v”}, 2.3, [“another”, “array”]];

- 2 types: Single dimensional and Multi dimensional (array of array(s)).

  i.e., var array = [“1st”, “2nd”, “3rd”];

  var arrayM = [[“1st”], [“2nd”], [“3rd”]];

- Accessing values stored in array: We use what is called index which is some integer ranges from 0 to array’s length - 1.

Arrays

```
var name = []; // empty array
var name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element
                                                      JS
```

```
var ducks = ["Huey", "Dewey", "Louie"];
var stooges = []; // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
                                                      JS
```

Array methods

```js
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
                                                         JS
```

- array serves as many data structures: list, queue, stack, ...
- methods: `concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift`
  - push and pop add / remove from back
  - unshift and shift add / remove from front
  - shift and pop return the element that is removed

ACCESSING ARRAYS

- Accessing Single dimensional array:

  i.e., array[0]; // 1st

  array[1]; // 2nd

  array[2]; // You guess here.
- Accessing Multi dimensional array:

  i.e., arrayM[0]; // [“1st”]

  arrayM[0][0]; // 1st.

ASSOCIATIVE ARRAYS

- Associative Arrays: Kind of Java Maps. Remember the Key-Value pairs?

  i.e., var pincodes = [];

  pincodes[“khanpur”] = 110062;

  pincodes[“badarpur”] = 110044;

  pincodes[“saket”] = 110017;
- Accessing Associative arrays:

  i.e., pincodes[“saket”]; // 110017

  pincodes[“badarpur”]; // You guess here.
- Associative arrays are actually **Object**s!

CONTROLLING PROGRAM FLOW

- Conditions: Making decisions.
- if statement: Expects a boolean expression.

Expression is combination of values, variable references, function calls and operators that evaluates to some value.

- Some comparison operators that we can use are: **less than (<), greater than (>), less than equal to (<=), greater than equal to (>=), equals to (==), not equals to (!=), not (!)**
- Multiple conditions operators: **and (&&), or (||)**

IF STATEMENTS

- if-else statement: if condition is false then execute the else block.
- else-if statements: It’s not reverse of the if-else. It just says if condition false then check this (else-if) condition.
- How it looks like:

```
i.e., if (condition) {
          // Some true code
         }else if (condition) {
          // Some 2nd true code
         }else {
          // Some false code
         }
```

LOOPS

- Loops: Minimizing repetition
- while loop: Simplest loop. While condition is true run the code. Condition can be any expression.
- do-while loop: Runs at least once, as condition is evaluated after running the loop body. As always condition can be any expression.
- for loop: Succinct all the above!
- All the loops must consist three things: Incrementer, conditional expression, logic to increase the incrementer – so that condition eventually turns to false.

LOOPS FACES

- How these look like?:
- while loop: i.e., while (condition) {

```
                    // Run the code
        }
```

- do-while loop: i.e., do {

```
                    // Run the code
            }while (condition)
```

- for loop: i.e., for (declaration; condition; action) {

```
                    // Run the code
      }
```

FUNCTIONS: WRITING CODE FOR LATER

- If we want to re-run some code again and again from different places, then put that code in a function and call this function.
- Functions are like little packages of JavaScript code waiting to be called into action.
- Some predefined functions are **alert(), prompt() and confirm()**. These all are available on the top level **window** object.
- Functions can return values. Values returned by predefined **window** functions are: **undefined**, **user input string** or **empty string** or **null**, **true** or **false**.

MY FUNCTIONS

- Writing your own functions:

  i.e., function sayHi() {

  alert(“Hi”);

  }
- Calling functions: i.e., sayHi();

Parentheses are needed to call the functions.

ARGUMENTS: PASSING DATA TO FUNCTIONS

- Arguments or parameters: When a function expects something then it’s called a **parameters**. While we pass the expected data to a function then it’s called **arguments**.
- Declaring parameters:

```
i.e., function sayHi(name) {
        alert(“Hi ”+name);
        }
```

- Calling function with arguments:

```
i.e., sayHi(“Manvendra”);
```

- Functions can contain any number of parameters.

A SECRET ARRAY

- arguments array: Functions have one secret. They contain the **arguments** array. This array contains all the arguments passed to the function.

  i.e., function poll() {

```
var affirmative = arguments[0];
var negative = arguments[1];
}

// Calling the function
poll(“affirmative”, “negative”);
```

RETURN AND SCOPE

- Returning: As we know functions can return the values. We use **return** statement to return something.

  i.e., function sayHi(name) {

  return “Hi ”+ name;

  }
- Scope: Can be local or global. Ensure functions always declare variables using the var keyword or else they will look for it in global scope and will modify them.

I HAVE ANOTHER FACE

- Alternate function syntax:

```
i.e., var sayHi = function() {
        alert(“Hi”);
        }
```

OBJECTS

- Objects exist as a way of organizing **variables** and **functions** into logical groups. If we create objects to organize some variables and functions then the terminology changes slightly, now they are called **properties** and **methods** respectively.
- We have used the objects in this presentation. Where? Didn't you use arrays? **Array** is an object of JavaScript. Same array can be created as follows: i.e., var array = **new** Array(1, 2, 3);
- This is called instantiation of object using the **new** keyword.
- Built-in objects: **String, Date, Math, Boolean, Number, RegExp, Object, Array**

CREATE YOUR OWN OBJECTS

- We can create our own objects to encapsulate the data and logic. i.e.,

```
 var Person = new Object();
Person.name = “Manvendra SK”;
Person.age = 23;
Person.speakName = function() {
alert(“Hello, I’m ”+ this.name);
 };

Person.speakName();
```

ALTERNTAE SYNTAX

- Alternate objects syntax: i.e.,

```
 var Person = {
name: “Manvendra SK”,
age: 23,
speakName: function() {
        alert(“Hello, I’m ” + this.name);
}
 };

 Person.speakName();
```

It’s JSON (JavaScript Object Notation) syntax

CREATING REAL OBJECTS

- We can create objects those can be instantiated using the **new** keyword.

```
var Person = function(name, age) {
  this.name = name;
  this.age = age;
};

Person.prototype.speakName = function() {
  alert(“Hello, I’m ” + this.name);
};
```

USING REAL OBJECTS

```
var manvendra = new Person("Manvendra", 23);

manvendra.speakName();
```

- This method of creating objects is called Prototype pattern.