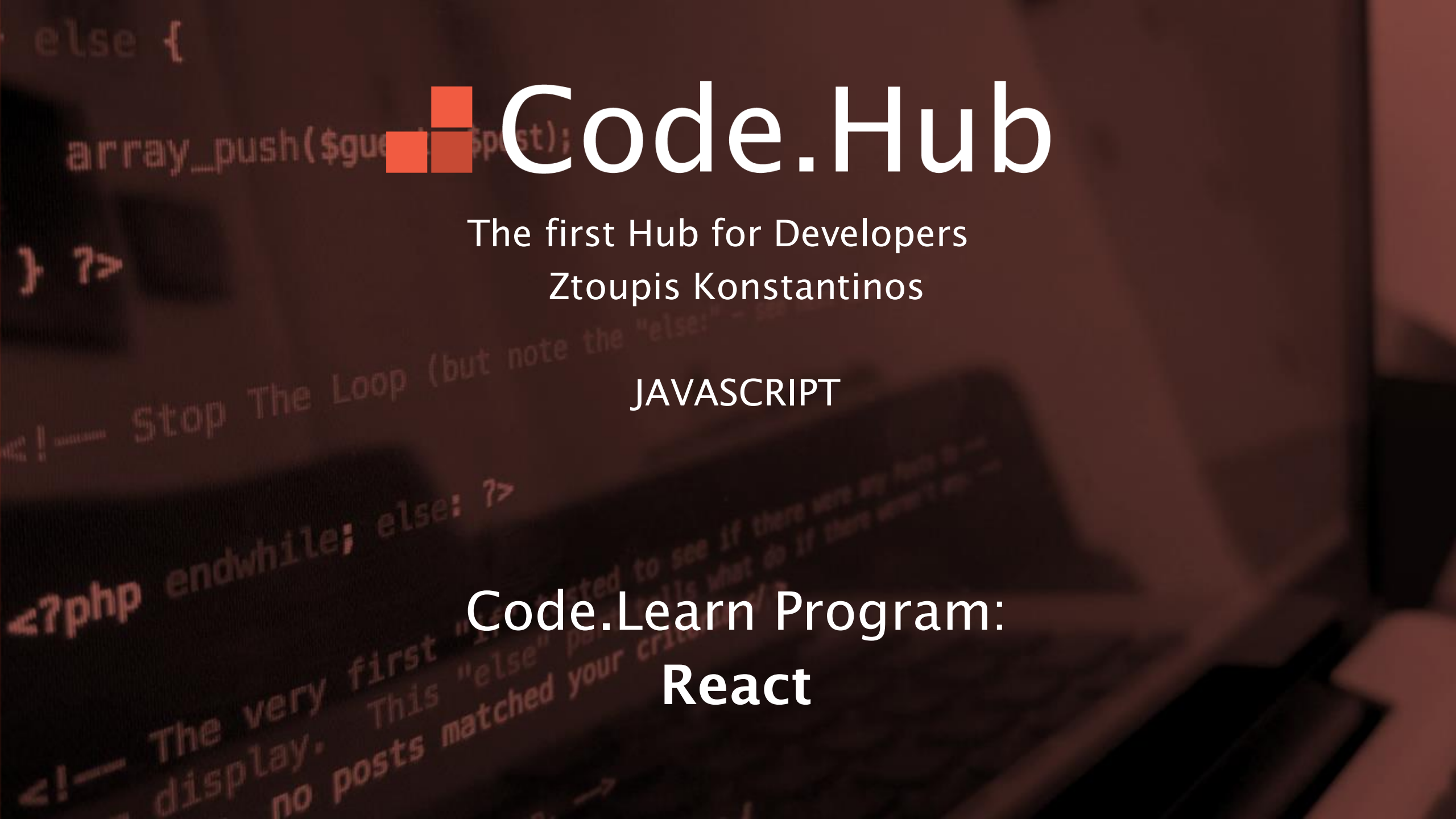


Code.Hub

The first Hub for Developers

Ztoupis Konstantinos

JAVASCRIPT

Code.Learn Program:
React

Recap

- HTML is the language used to describe the content and structure for a web page.
- CSS is the language used to describe the presentation of a web page. Makes stuff pretty!

JavaScript

- JavaScript is Netscape's cross-platform, object-based scripting language for client and server applications
- JavaScript is not Java. They are similar in some ways but fundamentally different in others.

JavaScript

JavaScript is the programming language used to manipulate the content of the webpage and make it interactive.

Makes things move!

JavaScript

- JavaScript is used in millions of web pages
  - to improve the design
  - to validate forms
  - to detect browsers
  - to create cookies
- JavaScript can react to events and can be used to validate data and to create cookies
- Is the most popular scripting language in all major browsers e.g.
  - Internet Explorer
  - Mozilla
  - Firefox
  - Netscape
  - Opera

JavaScript History

- Developed by Brendan Eich at Netscape, 1995
  - Scripting language for Navigator 2
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (aka JavaScript 1.5) -> ES5, …

Types of Scripting Languages

- Server-side Scripting Language
  - Can use huge resources of the server
  - Complete all processing in the server and send plain pages to the client
  - Reduces client-side computation overhead
- Client-side Scripting Language
  - Does not involve server processing
  - Complete application is downloaded to the client browser
  - Client browser executes it locally
  - Are normally used to add functionality to web pages e.g. different menu styles, graphic displays or dynamic advertisements

JavaScript and Java

JavaScript is NOT related to Java! They are two different programming languages.

- The JavaScript resembles Java but does not have Java's static typing and strong type checking.
- JavaScript supports most Java expression syntax and basic control-flow constructs.
- JavaScript has a simple, instance-based object model that still provides significant capabilities.

Embedding JavaScript in HTML

- By using the SCRIPT tag
- By specifying a file of JavaScript code
- By specifying a JavaScript expression as the value for an HTML attribute
- By using event handlers within certain other HTML tags

SCRIPT Tag

The <script> tag is an extension to HTML that can enclose any number of JavaScript statements as shown here:

```
<script>
    JavaScript statements...
</script>
```

A document can have multiple SCRIPT tags, and each can enclose any number of JavaScript statements

Hiding scripts in comment tags

```
<SCRIPT>
/* block comment */
JavaScript statements...
// single comment
</SCRIPT>
```

“Hello World”

```
<html>
<body>
  <script language="JavaScript">
    document.write(“Hello, World!”)
  </script>
</body>
</html>
```

JavaScript code in a file

- The SRC attribute of the <script> tag lets you specify a file as the JavaScript source (rather than embedding the JavaScript in the HTML).

- This attribute is especially useful for sharing functions among many different pages.

Why should I use external files

- Maintenance
- Easy to Read
- Performance (browser caching)
- Code separation

Putting JavaScript in HTML

```
    <script src="/static/js/index.js"></script>
</html>
```

Note: You want to load your scripts at the end of your HTML file!

Language syntax

- JavaScript is case sensitive
  - HTML is not case sensitive: onClick, ONCLICK, … are HTML
- Statements terminated by returns or semi-colons (;)
  - x = x+1; same as x = x+1
  - Semi-colons can be a good idea, to reduce errors
- “Blocks”
  - Group statements using { … }
- Variables
  - Define a variable using the var statement
  - Define implicitly by its first use, which must be an assignment
    - Implicit definition has global scope, even if it occurs in nested scope

Types

JavaScript has 5 primitive data types:

- Boolean
  - Two values: *true* and *false*
- Number
  - 64-bit floating point, similar to Java double and Double
  - No integer type
  - Special values *NaN* (not a number) and *Infinity*

Types

JavaScript has 5 primitive data types:

- String
  - Sequence of zero or more Unicode characters
  - No separate character type (just strings of length 1)
  - Literal strings using ' or " characters (must match)
- Null
- Undefined

Defining Variables

```
var myBoolean = true;

var myNumber = 5;

var mySecondNumber = 5.0;

var myString = "hello";
```

Types (null and undefined)

Types (null and undefined):

- null represents the intentional absence of any object value, is a type of object that is "nothing"

```
var myString = "home";

myString = null;
myString === null;
```

- undefined means unassigned

```
var unassignedVar;

unassignedVar === undefined;
```

Naming Variables

- 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 (for, function, if, in)

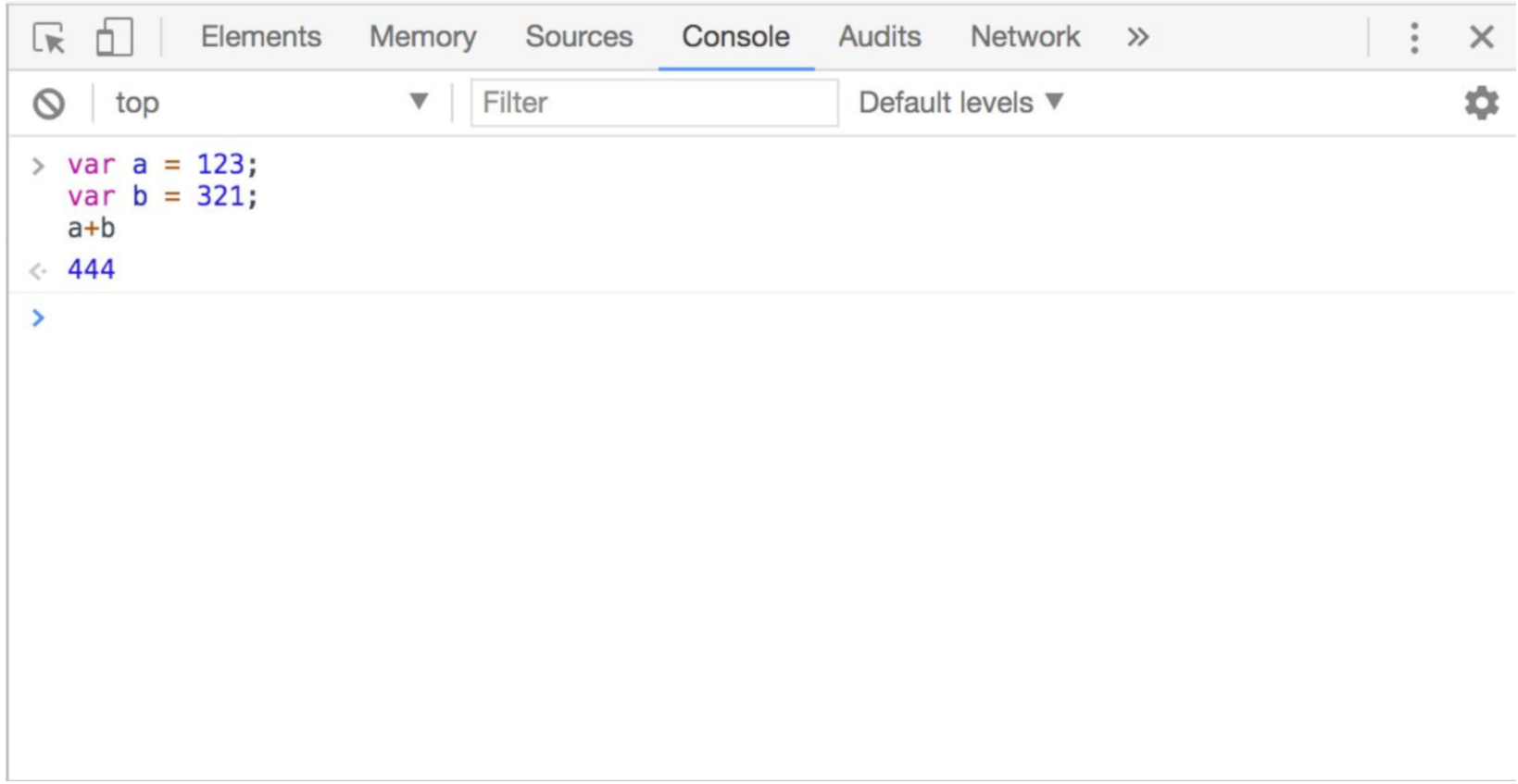
Variables Lifetime

- The **lifetime** of a JavaScript variable **starts when it is declared.**
- Local variables are deleted when the **function is completed**.
- In a web browser, global variables are deleted **when you close the browser window** (or tab).
- Function arguments (parameters) work **as local variables inside functions**.

Output

Console is your friend

console.log() will write to the JavaScript console



Code.Hub

Datatype conversion

- JavaScript is a loosely typed language:
  - not specify the data type of a variable when you declare it
  - data types are converted automatically as needed during script execution. ex: var answer = 42
- later, you could assign the same variable a string value, for example, answer = "thank you"

Datatype conversion

In expressions involving numeric and string values, JavaScript converts the numeric values to strings

```
var x = "The answer is " + 42
var y = 42 + " is the answer."
```

Operators

- Arithmetic Operators:
  - perform arithmetic operations between the values of the variables
  - Addition (+) , Subtraction (-),
  - Multiplication (*), Division (/), Modulus (%),
  - Increment (+ +), Decrement (- -)
- Assignment Operators:
  - assign values to variables
  - =, + =, - =, * =, / =, % =

Operators

- Comparison Operators:
  - determines equality or difference between variables or values
  - Equal to (= =), Exactly equal to (= = =),
  - Not equal (!=), Greater than (>), Less than (<),
  - Greater than or equal to (>=), Less than or equal to (<=)
- Logical Operators:
  - impose the logic between variables or values
  - AND (&&), OR (| |), NOT (!)

Operators

- Conditional Operator:
  - assign value to a variable based on some conditions
  - ?:

Operators

typeof Returns the type of a variable

instanceof Returns true if an object is an instance of an object type

- typeof "John“; // Returns "string“
- typeof 3.14; // Returns "number“
- cars instanceof Array; // Returns true
- cars instanceof Object; // Returns true

Conditional Statements

- **if statement**: to execute some code only if a specified condition is true
- **if...else statement**: to execute some code if the condition is true and another code if the condition is false
- **if...else if....else statement**: to select one of many blocks of code to be executed
- **switch statement**: to select one of many blocks of code to be executed

Looping

- JavaScript looping
  - Executes the same block of codes
  - Executes a specified number of times
  - Execution can be controlled by some control logic
  - uses **for**, **while**, **do….while** statements
  - uses **for**…**in** to iterate through the elements of an array
- **Break** breaks the loop and follows the code after the loop
- **Continue** breaks the loop and continues with next value.

Functions and Events

- JavaScript Functions
  - Can be called with the function name
  - Can also be executed by an event
  - Can have parameters and return statement
- Events
  - are actions that can be detected e.g. OnMouseOver, onMouseOut etc.
  - are normally associated with functions
  - <input type="text" size="30" id="email" onChange="checkEmail()">

Common Events

onchange:	An HTML element has been changed
onclick:	The user clicks an HTML element
onmouseover:	The user moves the mouse over an HTML element
onmouseout:	The user moves the mouse away from an HTML element
onkeydown:	The user pushes a keyboard key
onload:	The browser has finished loading the page

JavaScript and OOP

- JavaScript
  - is an Object Oriented Programming language
  - contains built-in JavaScript objects
    - String
    - Date
    - Array
    - Boolean
    - Math
    - RegExp
    - Window
    - Navigator
    - Screen
    - Location
    - History etc.

JavaScript and OOP

- also allows to define new objects
- objects contain Properties and Methods
- objects can be used as variable types

JavaScript: DOM

- To access the data in the HTML page
  - needs some data structures to access the HTML page
- Many browser implement an interface to what is called the Document Object Model (DOM)
  - It allows to output the document in the changed form to the browser
- DOM is a representation of the document in an object form, accessible from JavaScript programs

Strings Methods

- var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; txt.length; // 26
- txt[1]; // "B"
- txt.indexOf("D"); // 3
- txt.slice(7, 10); // "HIJ"
- txt.replace("BCD", "---"); // "A---EFGHIJKLMNOPQRSTUVWXYZ"
-
- "Hello World!".toUpperCase(); // "HELLO WORLD!"
-
- "Hello".concat(" ", " World!"); // "Hello World!"
- "Hello World!".split(' '); // ["Hello", "World!"]

Arrays

```
var animals = ['Tiger', 'Lion', 'Panther'];

animals.push('Cheetah'); // Tiger, Lion, Panther, Cheetah

animals.pop(); // Tiger, Lion, Panther
```

Accessing Arrays

```
var animals = ['Tiger', 'Lion', 'Panther'];

animals.push('Cheetah');

animals.pop();

consoLe.log(animals[1]) // Lion
```

Statements

Conditional Statement:
if…else

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

```
if (animals[0] === 'Zebra'
) {
  console.log("Zebra");
}
```

Loop Statements

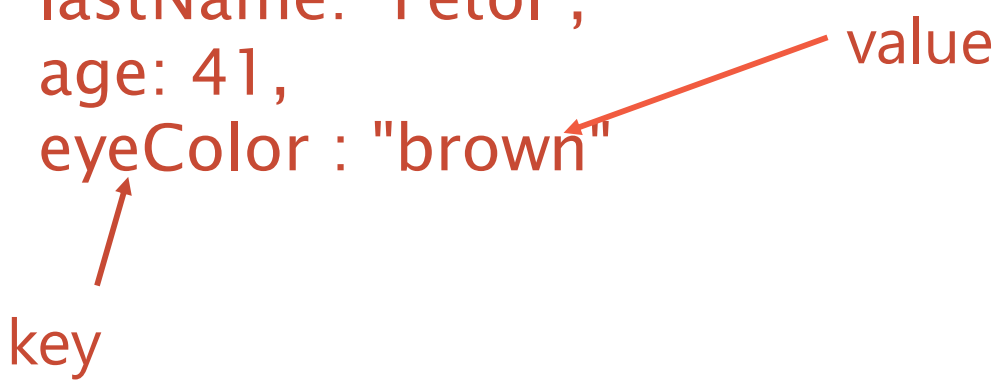
```
var animals = ['Tiger', 'Lion', 'Panther'];

for (var i = 0; i < animals.length; i++) {
  console.log(animals[i]);
}
```

Objects

A JavaScript object is a collection of key-value pairs.

```
var person = {
    firstName: "Paul",
    lastName: "Fetol",
    age: 41,
    eyeColor : "brown"
};
```



Objects (accessing fields)

```
var person = {
  firstName: "Paul",
  lastName: "Fetol",
  age: 41,
  eyeColor : "brown“
};

console. log(person.firstName); // Paul
console.log(person["firstName"]); //
Paul
console.log(person.age); // 41
console.log(person["eyeColor"]); //
brown
```

Objects and *this*

Property of the activation object for function call

- In most cases, *this* points to the object which has the function as a property (or “method”).

- Example :

```
var obj  = {
    x : 10,
    f : function(){return this.x}
}
obj.f(); // 10
```

Dates

- new Date()
- new Date(milliseconds)
- new Date(dateString)
- new Date(year, month, day, hours, minutes, seconds, milliseconds)

Dates Methods

getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Functions

Two ways to declare functions:

```
function getHello(name) {

  return name + " says hello!";

};

getHello("Jack");  // "Jack says hello!"
```

```
var getGreeting = function(name)
{

  return "Hi, " + name + "!";

};

getGreeting("Aaron"); //"Hi,
Aaron!"
```

Working with Functions

Variable Scope: defines where in the program a declared variable can be used

- Global variable
  - Declared outside a function and is available to all parts of the program
  - var keyword optional
- Local variable
  - Declared inside a function and is only available within the function it is declared
- Global and local variables can use same identifier

Checking Equality

ALWAYS use ===, DO NOT use ==

- === : value equality for primitives, reference equality for objects
- == : coerces the two things to be the same type first

```
2 === 2; // true

2 == 2;  // true
```

```
2 === "2";  // false

2 == "2";  // true
```

Checking Equality (===)

Value checks for primitives, reference checks for objects

```
var x = 2;

var y = 2;

x === y;  // true
```

```
var person1 = {name: 'Tom'};

var person2 = {name: 'Tom'};

person1 === person2 // ?
```

A quick tangent on references..

Variables are really references (point to something)

```
var person1 = {name: 'Tom'};
var person2 = {name: 'Tom'};
person1 === person2; // false
```

Regarding Semicolons...

They aren't strictly necessary, but you should always end each statement with a semicolon.

Callback Function

A callback function is a function that is passed as an argument to another function.

Example

```
var addTwo = function(x) {
  return x + 2;
};

var modifyArray = function(array, callback) {
  for (var i = 0; i < array.length; i++) {
    array[i] = callback(array[i]);
  }
};

var myArray = [5, 15, 25, 35]; // 5, 15, 25,
35
modifyArray(myArray, addTwo); // 7, 17, 27,
37
```

Anonymous functions

```
var modifyArray = function(array, callback)
{
  for (var i = 0; i < array.length; i++) {
    array[i] = callback(array[i]);
  }
};

var myArray = [5, 15, 25, 35]; // 5, 15,
25, 35
modifyArray(myArray, function(x) {
  return x + 2;
});
```

Good Practices

Conventions:

- Use === for equality checking
- End each statement with a semicolon
- camelCase your variables

Good Practices

Indenting:

- Use 2- or 4-space tabs, and be consistent

- And convert your tab characters into spaces

Playing with HTML

var el = document.getElementById('identifier');

Can represent elements from your HTML page in JavaScript

Modifying Elements in the HTML

```
var el = document.getElementById('identifier');
el.innerHTML = "I just put text in this element!";
```

Can change the content, styling, or structure of elements present in the HTML

Adding Elements to the HTML

```
var el = document.getElementById('identifierr');
var newEl = document.createElement('div');
newEl.innerHTML = 'Hi';
el.appendChild(newEl);
```

Can create new elements, and put them into the existing HTML

Event handlers

Events are actions that occur usually as a result of something the user does

- clicking a button is an event
- changing a text field
- moving the mouse over a hyperlink

You can define *event handlers*, such as **change and click**, to make your script react to events.

Giving Elements Actions to HTML

```
var button = document.getElementById('button');
button.addEventListener('click', function() {
 alert("Hi!");
});
```

Can find a button, give it an “action” every time someone clicks it

Putting it together!

```
var el = document.getElementById('identifier');
var button = document.getElementById('button');

button.addEventListener('click', function() {
  var newEl = document.createElement('div');
  newEl.innerHTML = 'Hi!';
  el.appendChild(newEl);
});
```

Find an element, find a button, add a listener that adds a new element to the first element whenever the button is clicked

In Conclusion

Can use JS to interact with html:

- add/remove divs
- get the contents of a textbox
- add/remove classes
- change CSS
- give elements actions/add listeners that do things on certain actions

Language features

- Stack memory management
  Parameters, local variables in activation records

- Garbage collection
  Automatic reclamation of inaccessible memory

- Closures
  Function together with environment (global variables)

- Exceptions
  Jump to previously declared location, passing values

- Object features
  Dynamic lookup, Encapsulation, Subtyping, Inheritance

- Concurrency
  Do more than one task at a time (JavaScript is single-threaded)