# PIC 40A

## Lecture 9: Introduction to JavaScript

# What is a scripting language?

- A scripting language is a lightweight programming language (simple syntax)
- scripting languages usually control an other software applications
- Scripts are written in different code from the application they control.
- Scripting languages are usually not compiled

# JavaScript

- Developed by Netscape in 1995
- Standardized in 1996 by European Computer Manufacturers Association (ECMA)
- Originally called Mocha then LiveScript and even known as ECMAScript but Sun allowed Netscape to use the name 'Java' in exchange for supporting Java in their browser
- JavaScript is officially a trademark of Oracle

# Some features of JavaScript

- It makes web pages interactive
- Makes it possible to react to events
- JavaScript can be used to create dynamic web pages
- With JavaScript it is possible to take absolute control of all elements on a page including style.
- JavaScript can be used to create web pages that act more like desktop applications.
- JavaScript can be used to validate data
- JavaScript can be used to create cookies
- JavaScript can be used to detect the visitor's browser

# Some features of JavaScript

- JavaScript  does not have to be compiled
- JavaScript can be embedded directly into HTML pages (or included as a separate file)
- JavaScript gives web designers a programming tool
- Everyone can use JavaScript without purchasing a license

# Hello World! Example

```
<html>
  <body>
    <script type="text/javascript">
      document.write("<h1>Hello World!</h1>");
    </script>
  </body>
</html>
```

# Placement of scripts

## Script inside <body> element

These scripts are executed immediately as the page loads

Use when you want your script to write page content as it loads.

# Scripts in <head> section

Scripts to be executed when they are called, or when an event is triggered, are placed in functions.

Put your functions in the head section, this way they are all in one place, and they do not interfere with page content.

## Put in the <head> section:

- Scripts that need to be executed before the browser starts loading the document body.
- Functions

# Scripts in <head> and <body>

You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

# Using an External JavaScript

- If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file.

- Save the external JavaScript file with a `.js` file extension.

**Note:** The external script cannot contain the `<script></script>` tags!

# External JS example

To use the external script, point to the `.js` file in the `src` attribute of the `<script>` tag:

```
<html>
    <head>
        <script type="text/javascript" src="myjavascript.js">
        </script>
    </head>
    <body>
    </body>
</html>
```

# "Script hiding"

You commonly see

```
<script type="text/javascript">
<!--
  // JavaScript here
//-->
</script>
```

Some browsers recognize script but do not have JS interpreter - These browsers automatically ignore the script.

Some browsers do not recognize the <script> tag and will show the contents of the script on the web page.

The comment also helps when validating - If the script includes XHTML tags such as <br/>, validation errors may occur.

# JavaScript Statements

A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do.

- Statements end with a semicolon

```
statement1;
statement2;
```

# JavaScript Identifiers

Variable names or function names

- Must begin with a letter, _, or $

Then can contain letters, _, $, or digits

- No length limitation
- Case sensitive
- By convention, programmer-defined names do not contain uppercase letters

# JavaScript reserved words

As with most languages there are many reserved words. Here is a brief list.

```
break, case, catch, continue,
default, delete, do, else, finally,
for, function, if, in, instanceof,
new, return, switch, this, throw,
try, typeof, var, void, while, with
```

# JavaScript variables

Variables in javascript are declared using the keyword var:

```
var x;
var person;
```

After the declaration shown above, the variables have value `undefined`

# Assigning values to variables

You can also assign values to the variables when you declare them:

```
var x=5;
var person_name="Philip J. Fry";
```

When you assign a text value to a variable use quotes around the value.

If you assign values to variables that have not yet been declared, the variables will automatically be declared.

```
x=5;
person_name="Philip J. Fry";
```

have the same effect as:

```
var x=5;
var person_name="Philip J. Fry";
```

# VARIABLES IN JAVASCRIPT HAVE IMPLICIT TYPE!!

JavaScript is an *untyped language*

JavaScript variable can hold data of any valid type

Variable takes its type from the data type of what it is holding

Variables can have the value of any primitive type or can be an object reference at any time

The following are legal:

```
var item;      // item is undefined
item = 10;     // item is now an integer
item = "ten"; // item is now a string
```

You cannot declare a type for variables in JavaScript

# JavaScript Primitive Types

A **primitive type** is built-in to the language, not composed of other types, and used as a basic building block to construct **composite** types.

Five Primitive Types of JavaScript

1. Number
2. String
3. Boolean
4. Undefined
5. Null

# JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

Given that **y=5**, the table below explains the arithmetic operators:

| Operator | Description | Example | Result |
|---|---|---|---|
| + | Addition | x=y+2 | x=7 |
| - | Subtraction | x=y-2 | x=3 |
| * | Multiplication | x=y*2 | x=10 |
| / | Division | x=y/2 | x=2.5 |
| % | Modulus (division remainder) | x=y%2 | x=1 |
| ++ | Increment | x=++y | x=6 |
| -- | Decrement | x=--y | x=4 |

# JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

Given that **x=10** and **y=5**, the table below explains the assignment operators:

| Operator | Example | Same As | Result |
|---|---|---|---|
| = | x=y | | x=5 |
| += | x+=y | x=x+y | x=15 |
| -= | x-=y | x=x-y | x=5 |
| *= | x*=y | x=x*y | x=50 |
| /= | x/=y | x=x/y | x=2 |
| %= | x%=y | x=x%y | x=0 |

# Numeric Literals

- All Number values are 64bit (8 bytes) floating point numbers

- Can have the form of
  - integers: `15, -41`
  - floating point values: 1.7, -9.3, 4.1e-3, -6E2
  - hexidecimal: `0x00F`
  - Values which cannot be interpreted as Numbers are given the value Not a Number: `NaN`

# String Literals

eg `"Hello!"`

- sequences of zero or more characters delimited by `''` or `" "`
- can include escape character sequences

e.g. `\n, \t, \', \", \\`

- null string `""` or `''`

# Concatenation

Just as in C++ the + can be used as a concatenation operator for strings

```
var first_name = "Philip";
var middle_initial = "J.";
var last_name="Fry";
var full_name = first_name + " " +
     middle_initial + " " last_name;
```

# Type convertions

- Iimplicit type convertions are Called coercions
- When a value of one type is used in an operation that requires a value of different type JavaScript implicitly changes the type
- Most commonly coercions happen between a string and a number.
- If either operand of + operator is a string the + is interpreted as concatenation.

# Examples

```
"November " + 2010
```

Evaluates to a string

So does

```
2010 + "November"
```

Consider

```
 7 * "3"
```

Since `*` does not make sense for strings the string `"3"` is attempted to be converted to a number. Here it succeeds and the expression has value `21`.

What about

```
7 * "November"
```

`"November"` cannot be converted to a number so it gets a value `NaN` and whole expression evaluates to `NaN`

# Explicit Type Conversions

Converting numbers to strings

Method #1

Use String constructor

```
var my_number=17;
var my_number_string = String(my_number);
```

# Explicit Type Conversions

Converting numbers to strings

Method #2

Use toString method

```
var my_number=6;
var my_number_string = my_number.toString(); // "6"
var my_number_binary_string = my_number.toString(2); // "110"
```

`toString(base)` allows you to choose the base in which the number is represented

# Explicit Type Conversions

Method #3

```
var my_number_string = my_number + "";
```

# Explicit Type Conversions: Converting strings to numbers

Method #1

Use the number constructor

```
var my_number = Number(my_number_string);
```

Method #2

```
var my_number = my_number_String - 0;
```

Limitations of converting strings to numbers:

The number cannot be followed by any character except a space

String must begin with a number literal or NaN is returned

# `typeof` Operator

Syntax:

`typeof( … )`

where … is either a variable or a literal expression.

Example:

```
var  true_or_false = true;
document.write(typeof(true_or_false));
```

`typeof` always returns a string

possible outputs are `"number"`, `"string"` , `"boolean"`, `"object"`, `"undefined"`

`"object"` is returned if the operand is an object or `null`

If the variable has not been initialized `"undefined"` is returned