

The background features a dark blue gradient with a series of curved, glowing lines that create a sense of depth and movement. On the right side, there is a prominent grid-like pattern that appears to be part of a larger, curved structure, possibly representing a tunnel or a futuristic architectural element. The overall aesthetic is clean, modern, and tech-oriented.

Scripting for Multimedia

LECTURE 3: INTRODUCING JAVASCRIPT

Understanding Javascript

- Javascript is not related to Java but to ECMAScript
- It is widely used for client-side scripting on the web
 - Javascript, Jscript, and ActionScript
- Javascript is untyped

Data

- Most data can be broken into smaller pieces called values.
- JS defines a value type as an **object**, a **primitive value**, or a **function**
 - Types of primitive value: *undefined*, *null*, *Boolean*, *number* or *string*
 - A *function* is a callable object
 - A *function* that is a member of an object is called a *method*
- Build-in objects in JS
 - the global object, the Object object, the Function object, the Array object, the String object, the Number object ...

Data

- Number type in JavaScript
 - double precision, 64-bit binary formation, IEEE 754 value

63 62...52 51...0
sign exponent fraction

- Special values
 - NaN
 - Infinity
 - -Infinity

Data

- You can create a **string** by encoding in **single** or **double** quotes
 - Examples
 - `"Every good boy deserves fudge"`
 - `'The quick brown fox jumps over the lazy dog'`
 - `'The doctor said "Today is your lucky day!"'`
 - `"I'm going to be happy when you give the news!"`
 - Using backslash (\)
 - Examples
 - `'The doctor said "I\'m pleased to announce that it\'s a girl!" '`
 - `"The doctor said \"I'm pleased to announce that it's a girl!\" "`
 - `\t` and `\n`

Data

- String concatenation using plus sign

- Examples

```
'Hickory Dickory Dock.' + "The mouse ran up the clock." + 'The  
clock struck one'
```

```
'Hickory Dickory Dock.' +  
"The mouse ran up the clock." +  
'The clock struck one'
```

Data

- Using unary operators
 - There is only a single operand
 - `typeof 'Hello World'`
 - `typeof 19.5`
 - `typeof true`

Data

- Boolean
 - `10 < 9; 20 > 3`
 - `5 <= 4; 7 >= 8`
 - `'Apples' != 'Oranges'`
 - `10 == 13-3`
- Logical operators
 - `'Apples' == 'Oranges' && 5 > 3`
 - `5 > 10 || 4 < 2`
 - `!(7 > 5 || 1 > 2)`
 - Short-circuiting operators

Statement

- Variables

- Examples

```
var totalCost = 2 * 21.15;  
var tax = totalCost * .05;
```

- or

```
var totalCost;  
var tax;  
totalCost = 3 * 21.15;  
tax = totalCost * 0.05;
```

Statement

- Rules for naming variables
 - A variable name can contain numbers but cannot begin with a number
 - 4YourEyes, 2give, 1ForAll X
 - Must not contain mathematical or logical operators
 - monday-friday, cost*5 X
 - Must not contain any punctuation marks other than `_` and `$`
 - Must not contain any spaces
 - Must not be JavaScript keywords
 - Case-sensitive

Statement

- The name of the variable should be descriptive enough

- Some examples

```
//bad examples
```

```
var last;           //last accessed date
```

```
var current;       //current vehicle
```

```
var changed;       //the vehicle make was changed
```

```
//good examples
```

```
var lastAccessedData;
```

```
var currentVehicle;
```


```
var vehicleMakeWasChanged
```

- Recommend to use camel casing

Function

- A **function** is a grouping of statements
 - A function can be declared by using the function keyword and then providing a name (**identifier**)
 - Example

```
function Add(x, y) {  
    return x + y;  
}  
//call function  
var a = 5;  
var b = 10;  
var c = Add(a, b);
```

A yellow vertical line is positioned to the right of the code. Two yellow curved arrows point from the line towards the function definition and its call. The top arrow points to the function definition block, and the bottom arrow points to the function call in the variable assignment.

Function

- A **function expression** produces a value of type function
 - Assign function expressions to variables or execute them directly
 - Example

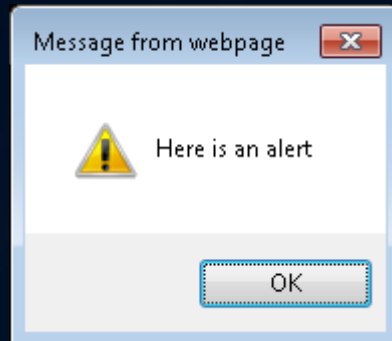
```
var addFunction = function(x, y) {  
    return x + y;  
};  
var c = addFunction(5, 10);
```
 - *addFunction* is called after the function expression assigned to *addFunction* variable
 - addFunction variable is of type **function**
 - An anonymous function

Function

- JavaScript is very loose when passing arguments to functions
 - Too many arguments → the extras is discarded
 - Arguments are not enough → parameter values for missing arguments will be undefined
- Advantage
 - You can add parameters to method already been created and called
- Disadvantage
 - You may inadvertently pass an incorrect quantity of arguments with no indication of a problem

Function

- Using the browser's built-in alert, prompt, and confirm functions
 - alert
`alert('Here is an alert');`

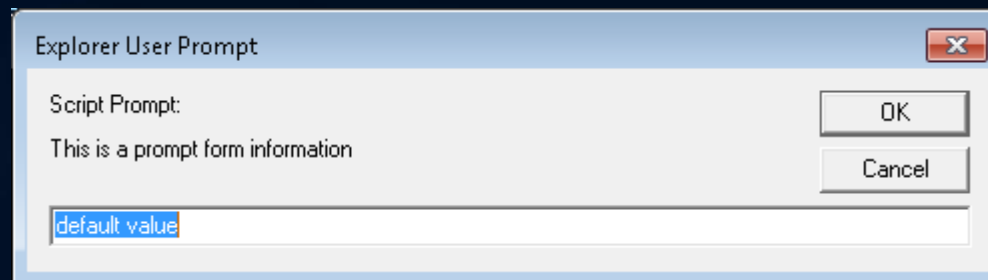


Function

- Using the browser's built-in alert, prompt, and confirm functions

- prompt

```
var prompResult = prompt('This is a prompt form  
information', 'default value');
```

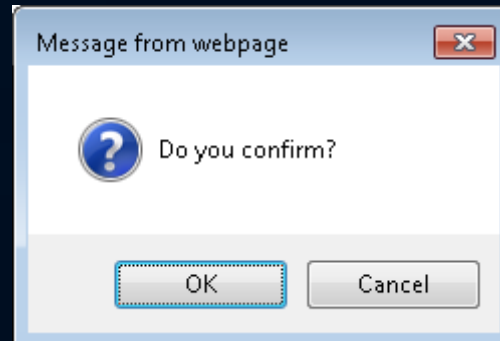


Function

- Using the browser's built-in alert, prompt, and confirm functions

- confirm

```
var confirmResult = confirm('Do you confirm?');
```



Function

- These built-in functions can be overwritten because the function name is variable

- Example

```
prompt = function(){  
    return 'hello again';  
}
```

Scoping variables

- In JavaScript, there are essentially two scopes
 - global
 - local
- Local scope is function scope
 - Variables declared *anywhere* inside the function will have a local function scope
 - Declare all function variables at the top of the function

Scoping variables

- Do **NOT** create global variables implicitly!

- Example

```
totalCost = 2 * 21.15;
```

```
tax = totalCost * .05;
```



Global? local?

Scoping variables

- Nesting functions

- Example

```
function areaOfPizzaSlice(diameter, slicesPerPizza) {  
    return areaOfPizza(diameter) / slicesPerPizza;  
    function areaOfPizza(diameter) {  
        var radius = diameter / 2;  
        return 3.1415926 * radius * radius;  
    }  
}
```

- Nested functions are private to the parent function
 - Variables in the nested function are local
 - A nested function can access variables in parent function and grandparent function

Converting to a different type

- **Number** function

- Example

```
var age = prompt('Enter age', '');  
alert('You will soon be ' + age + 1 + ' years old!');
```

- What will be displayed when running the code?

```
var age = prompt('Enter age', '');  
alert('You will soon be ' + Number(age) + 1 + ' years old!');
```

```
var age = prompt('Enter age', '');  
alert('You will soon be ' + (Number(age) + 1) + ' years old!');
```

Converting to a different type

- **String** function

- Example

```
var x = 10;
var y = 20;
alert(x + y);
//-----
var x = 10;
var y = 20;
alert(String(x) + String(y));
```

Conditional programming

- **if/else**

- Examples

```
var age = prompt('Enter age', '');  
if(isNaN(age))  
    alert('You need to enter a valid number');  
else  
    alert('You will soon be ' + (Number(age) + 1) + ' years  
old!');
```


Conditional programming

- **if/else**

- Examples

```
var age = prompt('Enter age', '');
if(isNaN(age)) {
    alert('You need to enter a valid number');
}
else if(Number(age) >= 50) {
    alert('You're old! You will soon be ' + (Number(age) + 1) + ' years
old!');
}
else if(Number(age) <= 20) {
    alert('You're a baby! You will soon be ' + (Number(age) + 1) + ' years
old!');
}
else {
    alert('You will soon be ' + (Number(age) + 1) + ' years old!');
}
```

Conditional programming

- **switch**

- Example

```
var carColor = prompt('What color car would you like to buy?',  
    'white');  
switch (carColor) {  
    case 'red':  
        alert('Red is a fancy choice!');  
        break;  
    case 'white':  
        alert('White is in stock and you get a discount!');  
    default:  
        alert('The color: ' + carColor + ' is not known.');
```

```
};
```

Conditional programming

- **switch**

- Example

```
var age = prompt('Enter your age', '');
age = Number(age);
switch (true) {
  case isNaN(age):
    age = 0;
    alert('You need to enter a valid number');
    break;
  case (age >= 50):
    age = Number(age) + 1;
    alert('You're old! You will soon be ' + age + ' years old!');
    break;
  default:
    alert('You will soon be ' + (Number(age) + 1) + ' years old!');
    break;
};
```

Conditional programming

- Determining whether a variable has a value using **if** keyword
 - If the variable has a value → true
 - If the variable is *undefined* or *null* → false

- Example

```
if(myVar) {  
    alert('myVar has a value');  
}  
else {  
    alert('myVar does not has a value');  
}
```

Conditional programming

- No value coalescing operators

- Example

```
var customer = prompt('Please enter your name');  
alert('Hello ' + (customer || 'Valued Customer'));
```

- || operators can be chained

```
var customer = prompt('Please enter your name');  
var companyName = prompt('Please enter your company name');  
alert('Hello ' + (customer || companyName || 'Valued  
Customer'));
```

- && operator exhibits similar behavior but returns the first empty value

Conditional programming

- Determine two values have the same type and are equal

- True

```
null == undefined;  
false == 0;  
' ' == 0;  
'123' == 123;
```

- False

```
null === undefined  
false === 0;  
' ' === 0;  
'123' === 123;
```

Loops

- **while**

- Example

```
var x = 10;
while(x > 0) {
    x--;
    alert("The value of x is " + x);
}
```

Loops

- **do**

- Example

```
var retries = 0;
do {
    retries++;
    showLoginScreen();
} while(!authenticated() && retries < 3);
if(retries==3) {
    alert('Too many tries');
    return;
}
```


Loops

- **for**

- Example

```
for(var counter = 0; counter < 10; counter++) {  
    alert('The counter is now set to ' + counter);  
}
```

Loops

- Breaking out of a loop (**break**)
 - break will exit only from the current loop

- Example

```
var numberToTest = prompt('Type number here.', '');
var index = 2;
var isPrime = true;
while (index < numberToTest) {
    if (numberToTest % index == 0) {
        isPrime = false;
        break;
    }
    index++;
}
```

Handling errors

- try/catch/finally

- finally block is executed after the try block successfully completes or the catch block completes.

- Example

```
try {  
    undefinedFunction();  
    alert('Made it, so undefinedFunction exists');  
}  
catch(ex) {  
    alert('The following error occurred: ' + ex.message);  
}  
finally {  
    alert('Finally block executed');  
}
```