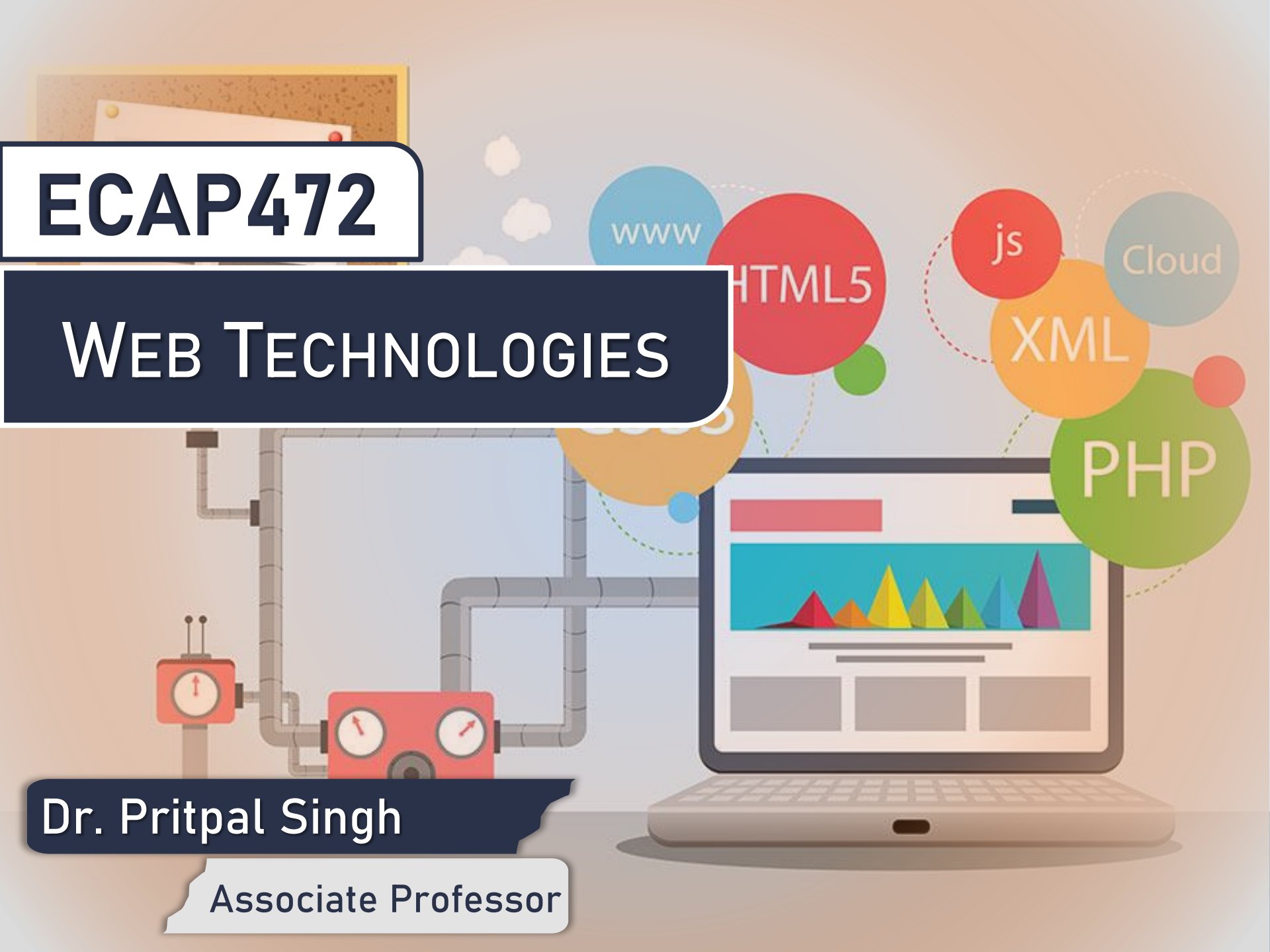
ECAP472

Web Technologies



Dr. Pritpal Singh

Associate Professor

Learning Outcomes

After this lecture, you will be able to

- understand concept of JavaScript functions,
- understand concept of conditional statements.

JavaScript Function

- JavaScript functions are defined with the function keyword.
- You can use a function declaration or a function expression.
- function *functionName*(*parameters*)

```
{
 // code to be executed
}
```

- Declared functions are not executed immediately. They are "saved for later use", and will be executed later, when they are invoked (called upon).
- Example

Example

- function myFunction(a, b)
- {
- return a * b;
- }
- Semicolons are used to separate executable JavaScript statements.

  Since a function declaration is not an executable statement, it is not common to end it with a semicolon.

Function Expressions

- A JavaScript function can also be defined using an expression.
- A function expression can be stored in a variable:
- Example
- const x = function (a, b) {return a * b};
- After a function expression has been stored in a variable, the variable can be used as a function

The Function() Constructor

- As you have seen in the previous examples, JavaScript functions are defined with the function keyword.
- Functions can also be defined with a built-in JavaScript function constructor called Function().

The Function() Constructor

- Example
- const myFunction = new Function("a", "b", "return a * b");
- let x = myFunction(4, 3);

Function Hoisting

- Hoisting is JavaScript's default behavior of moving declarations to the top of the current scope.
- Hoisting applies to variable declarations and to function declarations.
- Because of this, JavaScript functions can be called before they are declared:
- myFunction(5);

Function Hoisting

```
function myFunction(y)
 {
  return y * y;
}
```

- Functions defined using an expression are not hoisted.

Self-Invoking Functions

- Function expressions can be made "self-invoking".
- A self-invoking expression is invoked (started) automatically, without being called.
- Function expressions will execute automatically if the expression is followed by ().
- You cannot self-invoke a function declaration.
- You have to add parentheses around the function to indicate that it is a function expression:

Self-Invoking Functions

- Example

```
(function () {

  let x = "Hello!!";  // I will invoke myself

})();
```

JavaScript Function Parameters

- Function parameters are the names listed in the function definition.
- Function arguments are the real values passed to (and received by) the function.
- Parameter Rules

JavaScript Function Parameters

- JavaScript function definitions do not specify data types for parameters.
- JavaScript functions do not perform type checking on the passed arguments.
- JavaScript functions do not check the number of arguments received.

Arguments are Passed by Value

- The parameters, in a function call, are the function's arguments.
- JavaScript arguments are passed by value: The function only gets to know the values, not the argument's locations.

Arguments are Passed by Value

- If a function changes an argument's value, it does not change the parameter's original value.
- Changes to arguments are not visible (reflected) outside the function.

Objects are Passed by Reference

- In JavaScript, object references are values.
- Because of this, objects will behave like they are passed by reference:

Objects are Passed by Reference

- If a function changes an object property, it changes the original value.
- Changes to object properties are visible (reflected) outside the function.

The JavaScript this Keyword

- In JavaScript, the this keyword refers to an object.
- Which object depends on how this is being invoked (used or called).
- The this keyword refers to different objects depending on how it is used

What is this?

- In an object method, this refers to the object.
- Alone, this refers to the global object.
- In a function, this refers to the global object.
- In a function, in strict mode, this is undefined.
- In an event, this refers to the element that received the event.
- Methods like call(), apply(), and bind() can refer this to any object

JavaScript Function call()

- Method Reuse
- With the call() method, you can write a method that can be used on different objects.
- In JavaScript all functions are object methods.
- If a function is not a method of a JavaScript object, it is a function of the global object

Example

- The example below creates an object with 3 properties, firstName, lastName, fullName.
- const myObject = {
- firstName:"John",
- lastName: "Doe",
- fullName: function () {
- return this.firstName + " " + this.lastName;
- }
- }
- // This will return "John Doe":
- myObject.fullName();
- In the example above, this refers to the person object.

The JavaScript call() Method

- The call() method is a predefined JavaScript method.
- It can be used to invoke (call) a method with an owner object as an argument (parameter).

This Example Calls the FullName Method of Person, Using it on Person1:

```
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
const person1 = {
  firstName:"John",
  lastName: "Doe"
}
const person2 = {
  firstName:"Mary",
  lastName: "Doe"
}

// This will return "John Doe":
person.fullName.call(person1);
```

JavaScript if Else and Else if

- Conditional statements are used to perform different actions based on different conditions.
- In JavaScript we have the following conditional statements:
- Use if to specify a block of code to be executed, if a specified condition is true

JavaScript if Else and Else if

- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed

The if Statement

- Use the if statement to specify a block of JavaScript code to be executed if a condition is true.
- Syntax

```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

Example

- Make a "Good day" greeting if the hour is less than 18:00:

```
if (hour < 18) {
  greeting = "Good day";
}
```

- The result of greeting will be:
- Good day

The Else Statement

- Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Example

- If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

- The result of greeting will be:
- Good day

The Else if Statement

- Use the else if statement to specify a new condition if the first condition is false.
- Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

Example

- If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
```

- The result of greeting will be:
- Good morning

That’s all for now…