Instructor Notes:

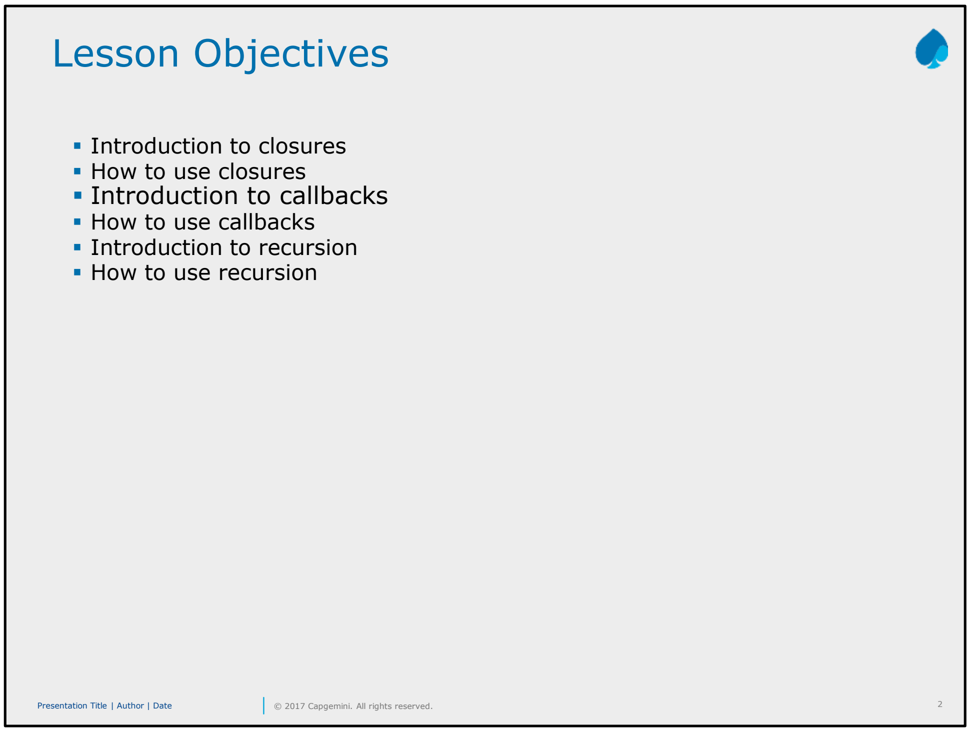
Add instructor notes here.

JavaScript ES6

Lesson 09- Closures, Callbacks, and Recursion

Instructor Notes:

Add instructor notes here.



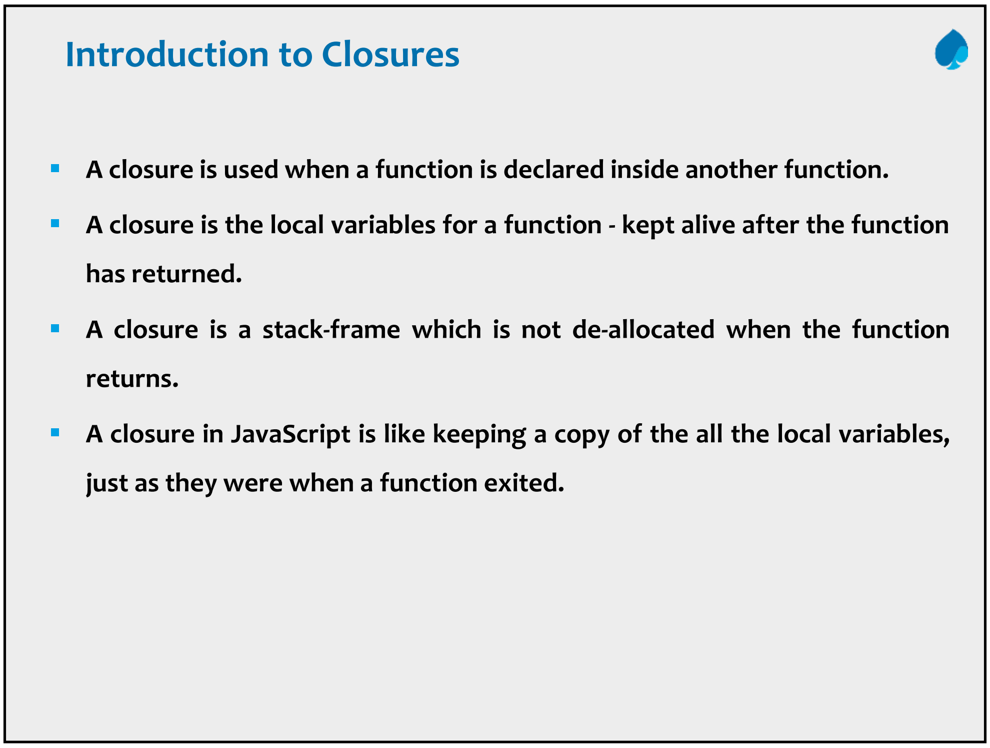
Lesson Objectives

- Introduction to closures
- How to use closures
- Introduction to callbacks
- How to use callbacks
- Introduction to recursion
- How to use recursion

Presentation Title | Author | Date © 2017 Capgemini. All rights reserved. 2

Instructor Notes:

Additional notes for instructor



Introduction to Closures

- **A closure is used when a function is declared inside another function.**
- **A closure is the local variables for a function - kept alive after the function has returned.**
- **A closure is a stack-frame which is not de-allocated when the function returns.**
- **A closure in JavaScript is like keeping a copy of the all the local variables, just as they were when a function exited.**

A closure is an inner function that has access to the outer (enclosing) function's variables—scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function's variables, and it has access to the global variables.
The inner function has access not only to the outer function's variables, but also to the outer function's parameters. Note that the inner function cannot call the outer function's *arguments* object, however, even though it can call the outer function's parameters directly.

Instructor Notes:

Additional notes for instructor

How to use closures

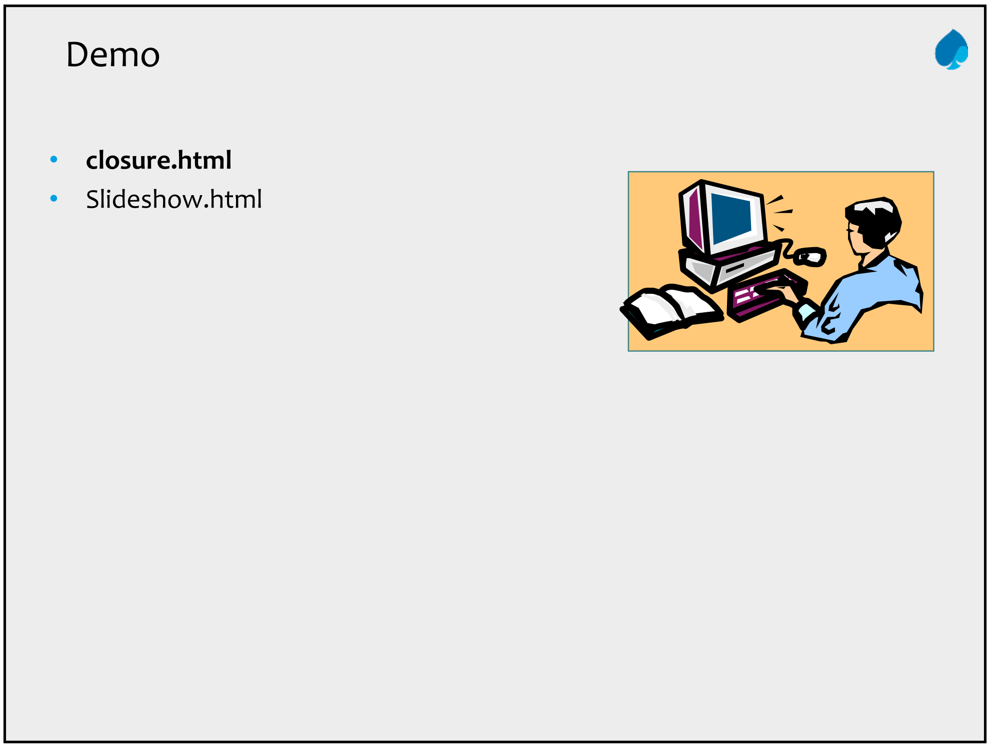
```
function showName (firstName, lastName) {
        var nameIntro = "Your name is ";
  // this inner function has access to the outer function's variables, including
the parameter
        function makeFullName () {
                return nameIntro + firstName + " " + lastName;
        }
return makeFullName ();
}
showName ("Simran", "Joshi"); // Your name is Simran Joshi
```

Closures' Rules and Side Effects

Closures have access to the outer function's variable even after the outer function returns:
Closures store references to the outer function's variables:
Closures Gone Awry
To fix this side effect (bug) in closures, you can use an **Immediately Invoked Function Expression** (IIFE)

Instructor Notes:

Additional notes for instructor



Demo

- **closure.html**
- Slideshow.html

Add the notes here.

Instructor Notes:

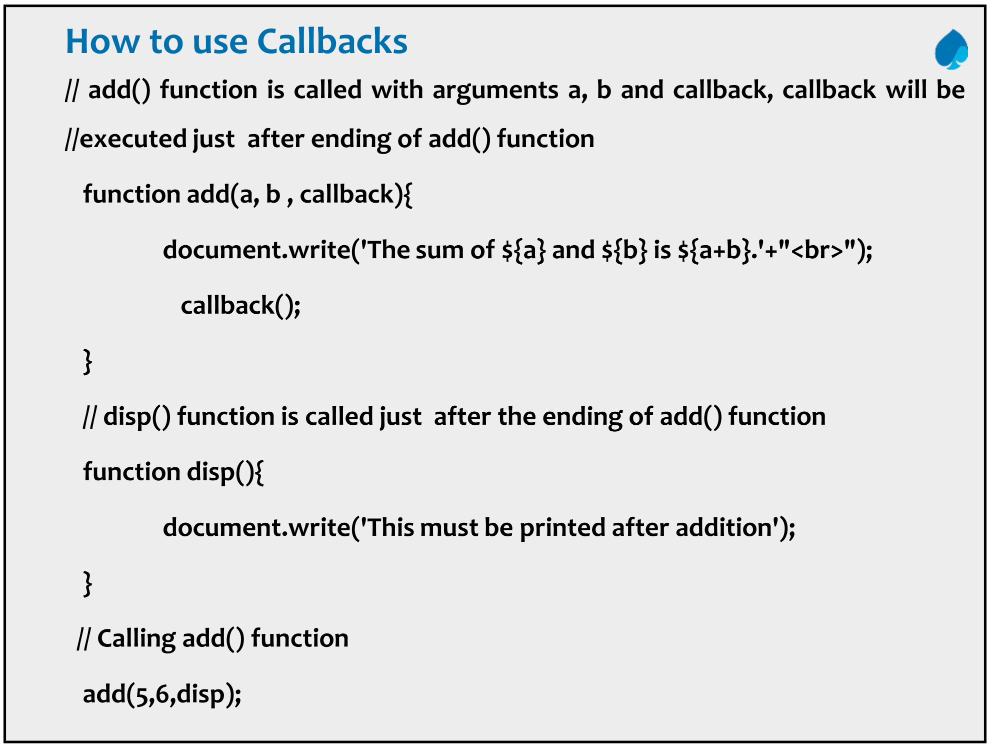
Additional notes for instructor

Introduction to Callbacks

- **Callback functions are derived from a programming paradigm known as functional programming.**
- **Functional programming specifies the use of functions as arguments**
- **A callback function, is a function that is passed to another function as a parameter.**
- **Callback function is called (or executed) inside the other Function.**
- **Callbacks are a great way to handle something after something else has been completed.**

Instructor Notes:

Additional notes for instructor



How to use Callbacks

```
// add() function is called with arguments a, b and callback, callback will be
//executed just  after ending of add() function
  function add(a, b , callback){
          document.write('The sum of ${a} and ${b} is ${a+b}.'+"<br>");
            callback();
  }
  // disp() function is called just  after the ending of add() function
  function disp(){
          document.write('This must be printed after addition');
  }
 // Calling add() function
  add(5,6,disp);
```

Output:
The sum of 5 and 6 is 11. This must be printed after addition

Explanation:
Here are the two functions – add(a, b, callback) and disp(). Here add() is called with the disp() function i.e. passed in as the third argument to the add function along with two numbers.
As a result, the add() is invoked with 1, 2 and the disp() which is the callback. The add() prints the addition of the two numbers and as soon as that is done, the callback function is fired! Consequently, we see whatever is inside the disp() as the output below the addition output.

Instructor Notes:

Additional notes for instructor

How to use Callbacks

```
// add() function is called with arguments a, b and callback, callback will be
//executed just  after ending of add() function
  function add(a, b , callback){
            document.write(`The sum of ${a} and ${b} is ${a+b}.` +"<br>");
              callback();
  }
   // add() function is called with arguments given below
  add(5,6,function disp(){
            document.write('This must be printed after addition.');
  });
```

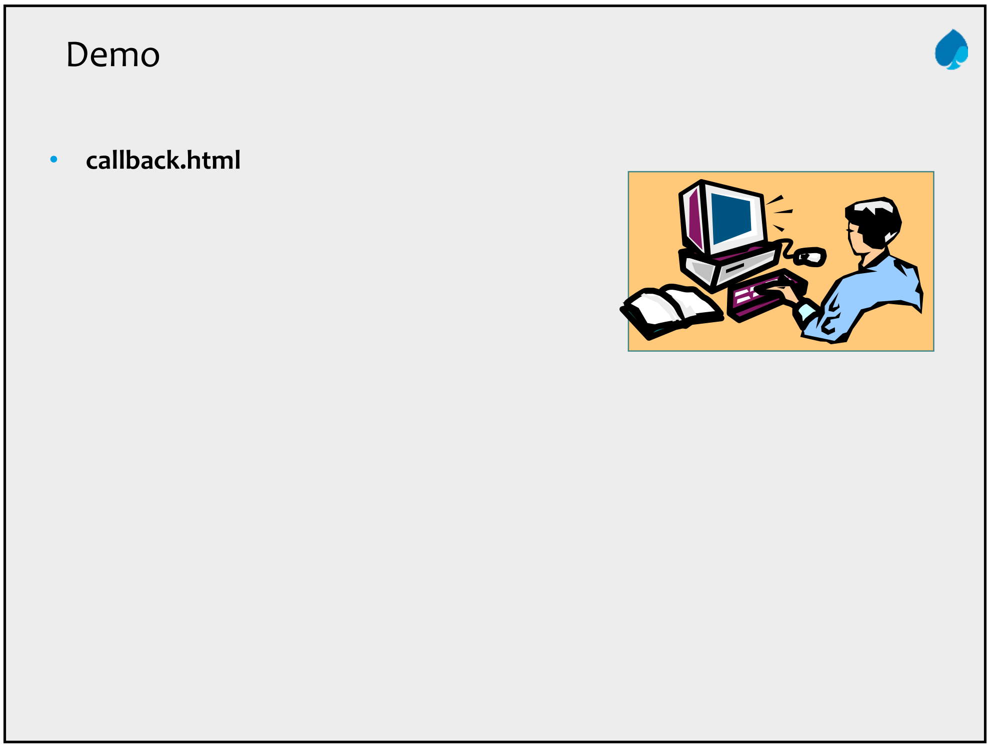
An alternate way to implement above code is shown below with anonymous functions being passed.

Output:

```
The sum of 5 and 6 is 11. This must be printed after addition
```

Instructor Notes:

Additional notes for instructor



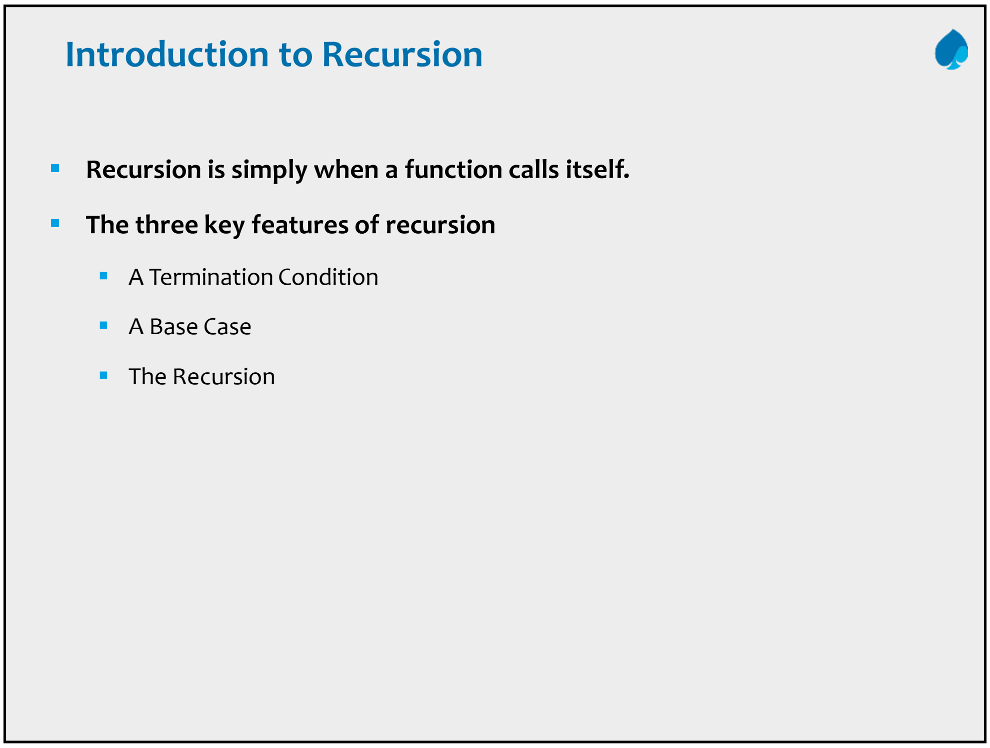
Demo

- **callback.html**

Add the notes here.

Instructor Notes:

Additional notes for instructor



Introduction to Recursion

- **Recursion is simply when a function calls itself.**
- **The three key features of recursion**
  - A Termination Condition
  - A Base Case
  - The Recursion

Instructor Notes:

Additional notes for instructor

How to use Recursion

```
function factorial(x) {
        if (x < 0) return;    // Termination Condition
        if (x === 0) return 1; //Base Case
        return x * factorial(x - 1);    // Recursion
}

factorial(3);
// 6
```

factorial(0) returns 1
factorial(1) returns 1 * factorial(0), or just 1*1
factorial(2) returns 2 * factorial(1), or just 2*1*1
factorial(3) returns 3 * factorial(2), or just 3*2*1*1

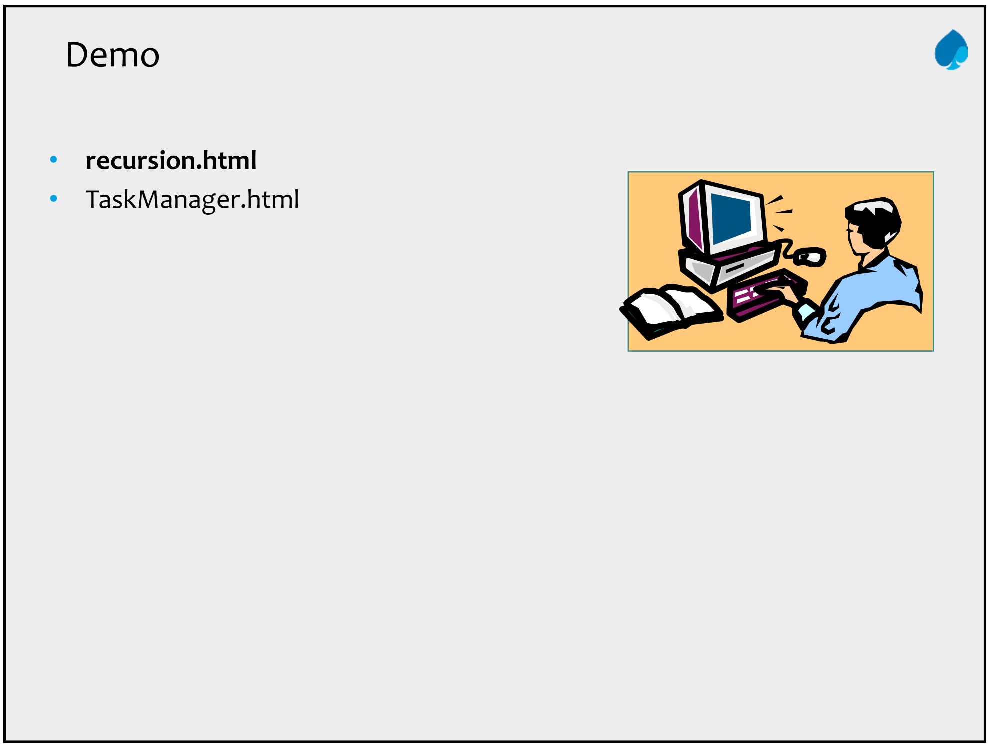
return 1 * 1 * 2 * 3
// 6
--
Here is the same explanation structured differently:
factorial(3) returns 3 * factorial(2)
factorial(2) returns 2 * factorial(1)
factorial(1) returns 1 * factorial(0)
factorial(0) returns 1

second example

```
function revStr(str){
if (str === '') return '';
return revStr(str.substr(1)) + str[0];
}
revStr('cat');
// tac
```

Instructor Notes:

Additional notes for instructor



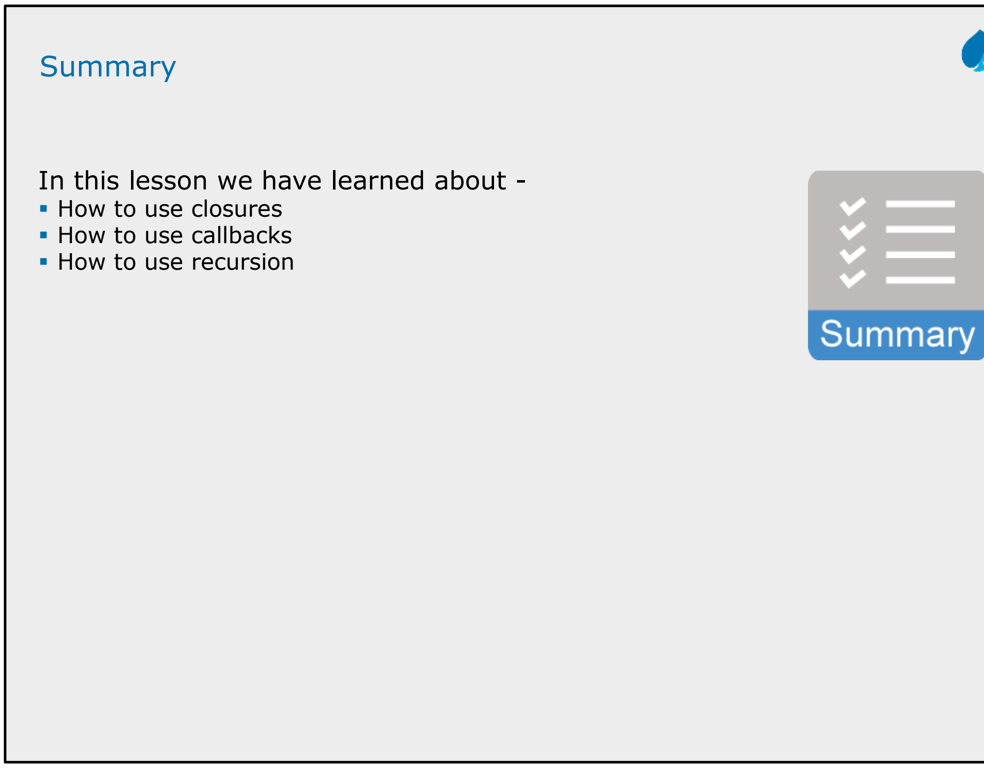
Demo

- **recursion.html**
- TaskManager.html

Add the notes here.

Instructor Notes:

Add instructor notes here.



Summary

In this lesson we have learned about -

- How to use closures
- How to use callbacks
- How to use recursion

Summary