Lecture Template

Function Expressions

Topics

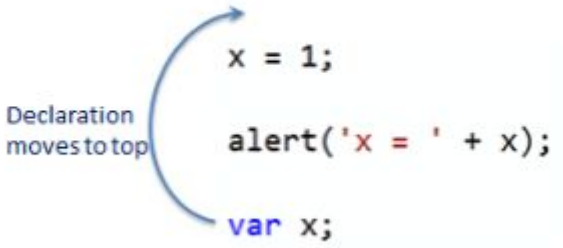
- Local vs Global Scope Recap
- Hoisting Recap
- Functional Expression
- Anonymous Functions

Video - Scope

Hoisting

Hoisting

- In most programming languages variables and functions need to be declared first, before you use them.
- Hoisting is a Js mechanism where variables and function declarations are moved to the top of their scope before code executes.
- This part is done when the program is being compiled (before the execution part).
- The code you write will remain the same, however the position of the declarations may change when compiled.



Function Expressions

Function Expressions

The **function** keyword can be used to define a function inside an expression.

You can also define functions using the Function constructor and a function declaration.

```
const getRectArea = function(width, height) {
  return width * height;
};

console.log(getRectArea(3, 4));
// expected output: 12
```

Function Expressions cont.

- in JavaScript, you can also store functions in variables. When a function is stored *inside* a variable it's called a **function expression**.
- Notice how the `function` keyword no longer has a name.

```
var catSays = function(max) {
  var catMessage = "";
  for (var i = 0; i < max; i++) {
    catMessage += "meow ";
  }
  return catMessage;
};
```

- It's an **anonymous function**, a function with no name, and you've stored it in a variable called `catSays`.
- And, if you try accessing the value of the variable `catSays`, you'll even see the function returned back to you.

```
catSays;
```

Returns:

```
function(max) {
  var catMessage = ""
  for (var i = 0; i < max; i++) {
    catMessage += "meow ";
  }
  return catMessage;
}
```

Function Expressions and Hoisting

- Deciding when to use a function expression and when to use a function declaration can depend on a few things, and you will see some ways to use them in the next section. But, one thing you'll want to be careful of is hoisting.
- All *function declarations are hoisted* and loaded before the script is actually run. *Function expressions are not hoisted*, since they involve variable assignment, and only variable declarations are hoisted. The function expression will not be loaded until the interpreter reaches it in the script.

- Being able to store a function in a variable makes it really simple to pass the function into another function. A function that is passed into another function is called a **callback**.

```
// function expression catSays
var catSays = function(max) {
  var catMessage = "";
  for (var i = 0; i < max; i++) {
    catMessage += "meow ";
  }
  return catMessage;
};
```

```
// function declaration helloCat accepting a callback
function helloCat(callbackFunc) {
  return "Hello " + callbackFunc(3);
}

// pass in catSays as a callback function
helloCat(catSays);
```

Functions As Parameters

- A function expression is when a function is assigned to a variable. And, in JavaScript, this can also happen when you pass a function *inline* as an argument to another function.

```
var favoriteMovie = function displayFavorite(movieName) {
  console.log("My favorite movie is " + movieName);
};

// Function declaration that has two parameters: a function for displaying
// a message, along with a name of a movie
function movies(messageFunction, name) {
  messageFunction(name);
}

// Call the movies function, pass in the favoriteMovie function and name of
movies(favoriteMovie, "Finding Nemo");
```

Inline Functions

- But you could have bypassed the first assignment of the function, by passing the function to the `movies()` function inline.

```
// Function declaration that takes in two arguments: a function for displayin
// a message, along with a name of a movie
function movies(messageFunction, name) {
  messageFunction(name);
}

// Call the movies function, pass in the function and name of movie
movies(function displayFavorite(movieName) {
  console.log("My favorite movie is " + movieName);
}, "Finding Nemo");
```

Inline Functions - Example 2

Why use anonymous inline function expressions?

- Using an anonymous inline function expression might seem like a very not-useful thing at first. Why define a function that can only be used once and you can't even call it by name?

- Anonymous inline function expressions are often used with function callbacks that are probably not going to be reused elsewhere. Yes, you could store the function in a variable, give it a name, and pass it in like you saw in the examples above. However, when you know the function is not going to be reused, it could save you many lines of code to just define it inline.

Function Expression Recap

- **Function Expression**: When a function is assigned to a variable. The function can be named, or anonymous. Use the variable name to call a function defined in a function expression.

```
// anonymous function expression
var doSomething = function(y) {
  return y + 1;
};
```

```
// named function expression
var doSomething = function addOne(y) {
  return y + 1;
};
```

```
// for either of the definitions above, call the function like this:
doSomething(5);
```

> *Returns:* 6

Function Expression Recap cont..

- You can even pass a function into another function *inline*. This pattern is commonly used in JavaScript, and can be helpful streamlining your code.

```
// function declaration that takes in two arguments: a function for displaying
// a message, along with a name of a movie
function movies(messageFunction, name) {
  messageFunction(name);
}

// call the movies function, pass in the function and name of movie
movies(function displayFavorite(movieName) {
  console.log("My favorite movie is " + movieName);
}, "Finding Nemo");
```

Video - Function Expressions