App Development & Modelling

BSc in Applied Computing

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

http://elearning.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

Functions & Scope

Creating Functions

- Functions contain blocks of code that need to be executed repeatedly.
- Functions can take zero or more arguments, and can optionally return a value.
- Functions can be created in a variety of way

```
// Function Declaration
function foo()
{
  /* do something */
}
```

```
// Named Function Expression
var foo = function()
{
  /* do something */
}
```

Using Functions

```
// A simple function
var greet = function(person, greeting)
{
  var text = greeting + ", " + person;
  console.log(text);
};

greet("Rebecca", "Hello");
```

```
// A function that returns a value
var greet = function(person, greeting)
{
  var text = greeting + ", " + person;
  return text;
};

console.log(greet("Rebecca", "hello")); // "hello, Rebecca"
```

Functions creating Functions

- the greet function returns a function!
- This functions is then called.

```
// A function that returns another function
var greet = function(person, greeting)
{
  var text = greeting + ", " + person;
  return function()
  {
    console.log(text);
  };
};

var greeting = greet("Rebecca", "Hello");
greeting();
```

Immediately-Invoked Function Expression (IIFE)

- A common pattern in JavaScript is the immediately-invoked function expression.

- This pattern creates a function expression and then immediately executes the function.

- This pattern is extremely useful for cases where you want to avoid polluting the global namespace with code — no variables declared inside of the function are visible outside of it.

```
// An immediately-invoked function expression
(function() {

  var foo = "Hello world";

})();

console.log( foo );   // undefined!
```

Functions as Arguments

- In JavaScript, functions are "first-class citizens" — they can be assigned to variables or passed to other functions as arguments.

- Challenging and difficult to read code!

```
// Passing an anonymous function as an argument
var myFn = function(fn)
{
  var result = fn();
  console.log(result);
};

// logs "hello world"
myFn(function()
{
  return "hello world";
});
```

Scope

- "Scope" refers to the variables that are available to a piece of code at a given time.

- A lack of understanding of scope can lead to frustrating debugging experiences.

- When a variable is declared inside of a function using the var keyword, it is only available to code inside of that function — code outside of that function cannot access the variable.

- On the other hand, functions defined inside that function will have access to to the declared variable.

More Scope...

- Furthermore, variables that are declared inside a function without the var keyword are not local to the function — JavaScript will traverse the scope chain all the way up to the window scope to find where the variable was previously defined.

- If the variable wasn't previously defined, it will be defined in the global scope, which can have unexpected consequences.

Scope Example 1

```
// Functions have access to variables defined in the same scope
var foo = "hello";
var sayHello = function()
{
  console.log(foo);
};

sayHello(); // "hello"
console.log(foo); // "hello"
```

Scope Example 2

```
// Code outside the scope in which a variable was defined does not have access
// to the variable
var sayHello = function()
{
  var foo = "hello";
  console.log(foo);
};

sayHello(); // hello

console.log(foo); // undefined
```

Scope Example 3

```
// Variables with the same name can exist in different scopes with different
// values
var foo = "world";

var sayHello = function()
{
  var foo = "hello";
  console.log(foo);
};

sayHello(); // logs "hello"
console.log(foo); // logs "world"
```