**Functions**

Functions are first class citizens in JavaScript. A function is an object, and therefore a value and can be used like any other value.

For instance a function can be returned from a function:

**function factory () {  
  return function doSomething () {}  
}**

A function can be passed to another function as an argument:

**setTimeout(function () { console.log('hello from the future') }, 100)**

A function can be assigned to an object:

**const obj = { id: 999, fn: function () { console.log(this.id) } }  
obj.fn() // prints 999**

When a function is assigned to an object, when the implicit **this** keyword is accessed within that function it will refer to the object on which the function was called. This is why **obj.fn()** outputs **999**.

It's crucial to understand that **this** refers to the object on which the function was called, not the object which the function was assigned to:

**const obj = { id: 999, fn: function () { console.log(this.id) } }  
const obj2 = { id: 2, fn: obj.fn }  
obj2.fn() // prints 2  
obj.fn() // prints 999**

Both **obj** and **obj2** to reference the same function but on each invocation the **this** context changes to the object on which that function was called.

Functions have a **call** method that can be used to set their **this** context:

**function fn() { console.log(this.id) }  
const obj = { id: 999 }  
const obj2 = { id: 2 }  
fn.call(obj2) // prints 2  
fn.call(obj) // prints 999  
fn.call({id: ':)'}) // prints :)**

In this case the **fn** function wasn't assigned to any of the objects, **this** was set dynamically via the **call** function.

There are also fat arrow functions, also known as lambda functions:

**const add = (a, b) => a + 1  
const cube = (n) => {  
  return Math.pow(n, 3)  
}**

When defined without curly braces, the expression following the fat arrow (**=>**) is the return value of the function. Lambda functions do not have their own **this** context, when **this** is referenced inside a function, it refers to the **this** of the nearest parent non-lambda function.

**function fn() {  
  return (offset) => {  
   console.log(this.id + offset)  
  }  
}  
const obj = { id: 999 }  
const offsetter = fn.call(obj)  
offsetter(1) // prints 1000 (999 + 1)**

While normal functions have a **prototype** property (which will be discussed in detail shortly), fat arrow functions do not:

**function normalFunction () { }  
const fatArrowFunction = () => {}  
console.log(typeof normalFunction.prototype) // prints 'object'  
console.log(typeof fatArrowFunction.prototype) // prints 'undefined'**