More on Objects and The "this" Keyword

A Light Intro to Object Oriented Programming

We'll cover the following

- Exercise
- this
- Object construction using this
  - Exercise

Let's go back to the `student` example we were using in earlier lessons.

```
var student = {
  name: "Mary",
  age: 10
};
```

In this example, each student has a `name` and an `age` property. If we wanted to create another student, we could define another object with the same properties:

```
var student2 = {
  name: "Michael",
  age: 12
};
```

However, if we had to define *many* student objects, having to write out the same properties over and over again will get tiring.

A better way to create a `student` object would be to **create a function** that returns an object:

```
var createStudent = function(name, age) {
  var student = {
    name: name,
    age: age
  }
  return student;
}

var student1 = createStudent("Mary", 10);
```

```
var student2 = createStudent("Michael", 12);

console.log("Students:", student1.name, student2.name);
```

Exercise

Use the `createStudent` function to create a new student, stored in a variable named `student3`. Make sure to give `student3` both `name` and `age` properties.

```
// write your code here
```

Now, we've *abstracted* the creation of a student into a more general function. This pattern is part of a computer science concept referred to as **object-oriented programming**.

The idea behind object-oriented programming is to represent data in **objects** as *properties*. We can then also define *functions* (or *methods*) that allow us to *modify* those properties. For instance, we could create functionality to increment the student's age on their birthday:

```
var birthday = function(student) {
  student.age++;
}

birthday(student1);
console.log(student1.age);
```

As with many other programming concepts, there is a better way to write the code above. Instead of defining a separate function that requires us to *pass in* the object as an argument, we could make the function **another property** of the object:

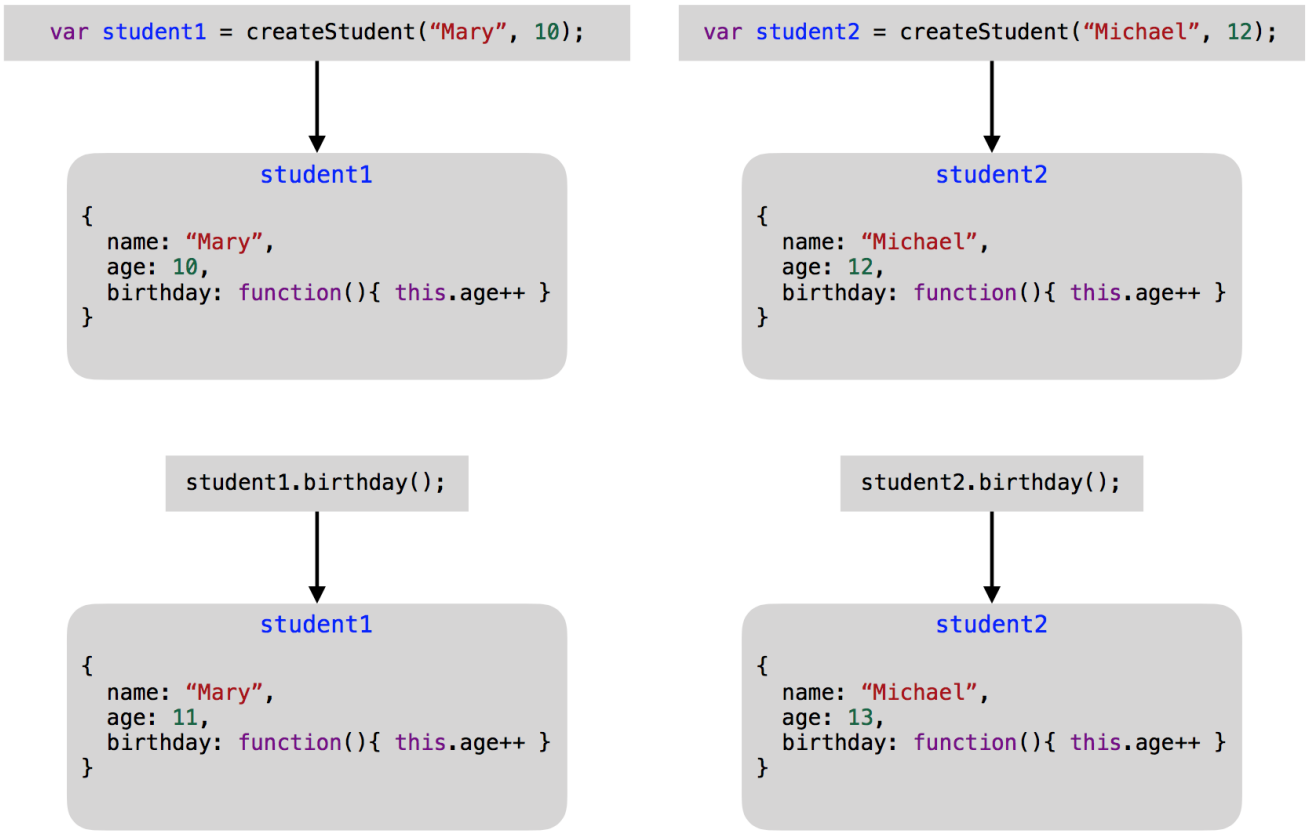
```
var createStudent = function(name, age) {
  var student = {
    name: name,
    age: age,
    birthday: function(){
     this.age++;
    }
  }
```

```
  return student;
}


var student1 = createStudent("Mary", 10);
var student2 = createStudent("Michael", 12);

student1.birthday();
console.log(student1.age)
```

`this`

The code above introduces a new concept: the `this` keyword. What exactly does `this` do?

Every time we call the `createStudent()` function, a **new instance** of a `student` object is created. The `this` keyword allows us to create functions that modify *the specific instance of the object* to which the function is attached.



'this' refers to the particular object instance you are working with

When we call the `birthday()` function, `this` takes a look at which object the function is residing in, and then accesses the `age` property from that object.

Object construction using `this`

Javascript has a specific syntax for **constructing** objects. Instead of *returning* an object, we can simply add properties to a function using `this`, like so:

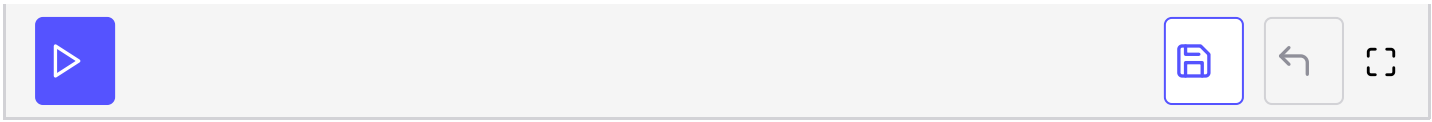
```
var Student = function(name, age) {
  this.name = name;
  this.age = age;
}
```

An object constructor function

When you use `this` to add properties to a function, Javascript treats the function as an **object constructor**, because it will instantiate (or *construct*) a new object with the properties defined using `this` whenever it is called.

To create a new object using the `Student` function, we can *explicitly* instantiate an object using the `new` keyword, like so:

```
var student1 = new Student("Mary", 11);

console.log("Student 1: " + student1.name);
```

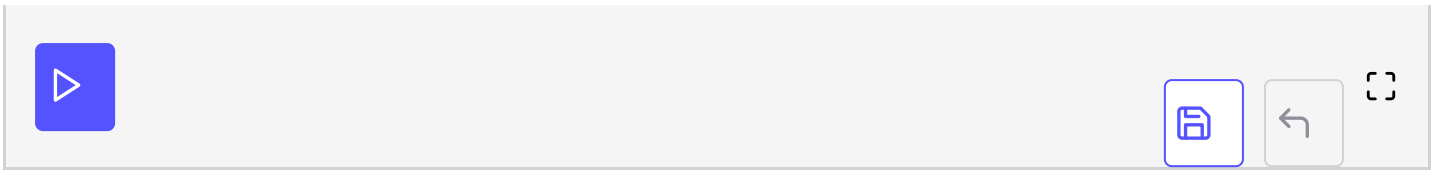


With object constructors, the intent of our code becomes much clearer. Every time we use the `new` keyword with the `Student` function, we are creating a new `student` object.

It should be noted that it is standard practice to start constructor names with a capital letter, so that it is clear that the function is an object constructor.

We can also define *methods* using the constructor syntax:

```
var Student = function(name, age) {
  this.name = name;
  this.age = age;
  this.birthday = function(){
   this.age++;
  }
}

var student1 = new Student("Mary", 11);
student1.birthday();


console.log("Student 1 Age:" + student1.age);
```

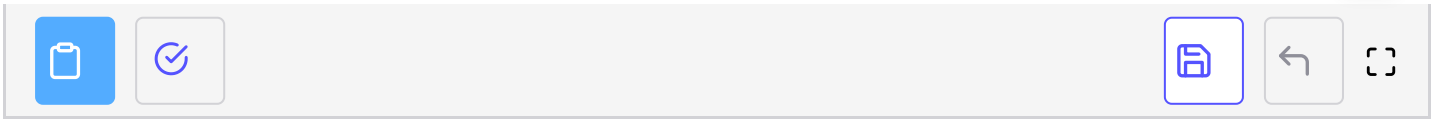


While the intent of creating objects and using the `this` keyword may not be clear as of yet, it will be super helpful to have an understanding of these concepts when we start delving deeper into DOM manipulation.

Exercise

Use the `Student` function to **construct** a *new* student object. Store this object in a variable named `student4`. Make sure to give `student4` both `name` and `age` properties.

```
// write your code here
```



Until now have learned about functions, conditional statements, loops, arrays, objects, this keyword in Javascript. Using your knowledge of javascript, let's learn to manipulate the HTML page using Javascript in the next chapter.