**Classes**

A recent update to JavaScript (ES6 in 2015) has introduced **classes** to the language. A **class** is a template that you can use to create an instance of an **object**. Think of it like a factory that contains blueprints for objects. You can supply it some parameters and the class will create an object according to those specifications. Before the introduction of classes you would have to use other methods in JavaScript to achieve the same goal. These include factory of generator functions. Below is how you can declare a class. Note that class declarations are not **hoisted** (unlike functions) so you must declare a class before trying to use it.

**Defining a Class**

**class HumanClass {** *// Note the first letter of class name should be uppercase*

**constructor( para1, para2 ) {** *//* ***constructor*** *is inbuilt method that creates properties for object*

**this.name = para1;** *// Parameters from* ***constructor*** *can be used to give values*

**this.age = para2;** *// to properties*

**this.occupation = "student";**

**}**

**}**

The **constructor** method should be a default in all classes and is called immediately to produce simple properties for the object.

**Creating Object from Class**

To use this class to create an object you would use the following syntax. Don’t forget the **new** keyword in front of the class name and the parameters are passed in the parenthesis **( )** at the end.

**const humanObject1 = new HumanClass( "James", 25 );**

**const humanObject2 = new HumanClass( "Mary", 30 );**

**const humanObject3 = new HumanClass( "Samantha", 19 );**

This creates three new objects which are similar. **humanObject1** looks like this if we created it in literal form:

**const humanObject1 = {**

**name: "James",**

**age: 25,**

**occupation: "student"**

**}**

**Object Methods in a Class**

You have seen how the **constructor** method above created properties for the objects. You can add methods to objects simply by writing functions, without the keyword **function**, in the class body but outside of the **constructor** method. Let’s demonstrate on the same class as before.

**class HumanClass {** *// Note the first letter of class name should be uppercase*

**constructor( para1, para2 ) {** *//* ***constructor*** *is inbuilt method that creates properties for object*

**this.name = para1;** *// Parameters from* ***constructor*** *can be used to give values*

**this.age = para2;** *// to properties*

**this.occupation = "student";**

**}** *// note there are no* ***,*** *commas here*

**introduction() {** *// note do not use the* ***function*** *keyword*

*// you can access the object’s properties using* ***this***

**alert( "My name is " + this.name );**

**}**

**calculateBirthYear( paraX ) {** *// methods can use parameters as well*

**return paraX - this.age;**

**}**

**}**

Let’s create another object from the class and call the methods.

**const newHumanObject = new HumanClass( "James", 25 );**

**newHumanObject.introduction();** *// alerts “My name is James”*

**newHumanObject.calculateBirthYear( 2021 );** *// returns 1996*

**Class Inheritance – A Class borrowing from another Class**

You’ve seen us create objects from a class. Classes can also borrow methods and properties from other existing classes. They do this using the **extends** keyword to select the parent Class you want to borrow from and the **super** method which allows us to call and use the parent Class’s **constructor** function. First you might be wondering why we would need this. Classes can be very general containing broad or wide-ranging properties. This might be well and good, but we might need to create objects which borrow some of these general details but also needs some more specific information relevant only to itself.

To describe this scenario, we want to create an object of a single McDonalds fast food store. Let’s use the McDonalds corporation as our parent class. It has a name “McDonalds”, advertising slogans, a brand and a menu that everyone is familiar with. Some, or all of this data can be useful to a single store. However, to create an effective object referencing the single store we’re going to need more specific details like an address, employees and maybe local menu items not found in other McDonalds stores. This data may not be useful to the corporation class. Here we should create a store class holding its own specific data/methods, but also borrows from the parent class.

*// the parent class is just a regular class*

**class ParentCorporation {**

**constructor( name ) {**

**this.name = name;**

**this.menu = ["Big Mac", "Fries", "Milkshake"];**

**this.slogan = "I’m loving it";**

**}**

**brandJingle() {**

**return "****Bah bah bah…" + this.slogan ;**

**}**

**}**

*//* ***extends*** *allows us to borrow from ParentCorporation*

**class Store extends ParentCorporation {**

*// the constructor function borrows* ***name*** *from parent*

*// thanks to* ***super()*** *method.* ***name*** *is passed in as parameter*

*// to the ParentCorporation class*

**constructor( address, localCuisine, name ) {**

**super( name );**

**this.address = address;**

**this.localCuisine = localCuisine;**

**}**

**greeting() {**

*// All properties and methods from ParentCorporation are usable*

*// in Store class. Notice* ***this.brandJingle()****,* ***this.name*** *and* ***this.menu***

**alert( this.brandJingle() + ". Welcome to " + this.address + " " + this.name );**

**const fullMenu = [ …this.menu ];** *// … look up js spread operator*

**fullMenu.push( …this.localCuisine );** *// merging two arrays for full menu*

**alert( "Full Menu: " + fullMenu.join( ", " ) );**

**}**

**}**

*// “address”, [ localCuisine ], “name”*

**const canadaMcD = new Store("Canada", ["Poutine"], "McDonalds");**

**const franceMcD = new Store("France", ["Wine", "Fondu"], "McDonalds");**

**const japanMcD = new Store("Japan", ["Sushi"], "McDonalds");**

\* Note if you copy and paste the above into the browser console, due to a pasting issue you may have to retype the **…** spread operator

Calling the **greeting()** method on the **franceMcD** object as shown below would produce 2 different alerts.

**franceMcD.greeting();**

**alert 1:** Bah bah bah…I’m loving it. Welcome to France McDonalds

**alert 2:** Full Menu: Big Mac, Fries, Milkshake, Wine, Fondu