Classes

JavaScript

Classes in Programming

We took a look at very simple objects that exist on their own, featuring properties. Those can feature methods as well, though methods are not JSON-compatible.

In this presentation we'll take a look at object blueprints, called classes.

Classes are found in most any object-oriented programming language, and are meant to act as guides you write for which properties and methods certain types of objects will have and use.

In JavaScript, you may have noticed that arrays are treated as objects in JavaScript. This means that there is a class that defines an array's properties and behaviours.

You may recall that arrays, for instance, always have a length property. We use it to see how many items are inside of the array.

If we want to add a new item to an array, we used the push method.

The class tells JavaScript that for an object to be an array, they have to follow those rules.

Properties and Methods

Properties are much like variables, in that they are names representing values.

They differ in that they are values held inside of an object, however; you will not see these values on their own like variables.

Methods are much like functions, in that they are names representing a set of instructions (callable code block.)

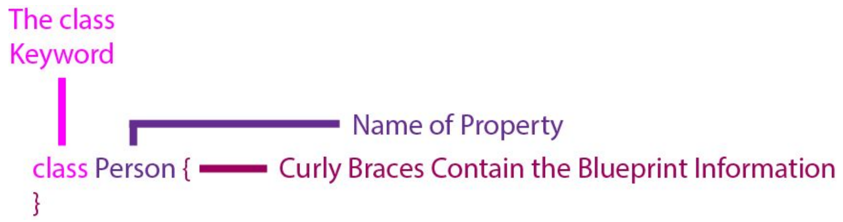
They are typically defined within an object's class, and are accessible to all objects following that blueprint.

Declaring a Class

Let's try writing our own simple blueprint. First thing's first!

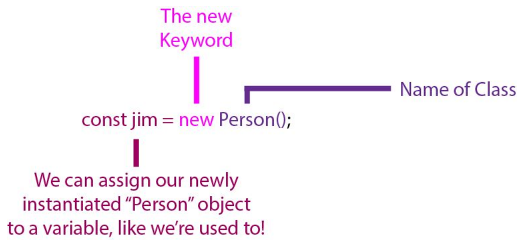
To declare a class, we use the class keyword.

Note that you can only declare a class with a specific name once in your program—declaring two classes of the same name will result in an error.



```
class Person {
}
```

Now, if you wanted to create an object out of this class, you use the new keyword followed by its name and parentheses…



```
const jim = new Person();
```

The new Keyword
Name of Class
We can assign our newly instantiated "Person" object to a variable, like we're used to!

Constructor Method

You may have noticed that there were parentheses after the class name, when we are creating new objects from the class. These aren't just for show!

They give us the opportunity to pass in arguments, much like with a function. There is a method inside of objects by default that is responsible for dealing with expected arguments.

This method is called the constructor.

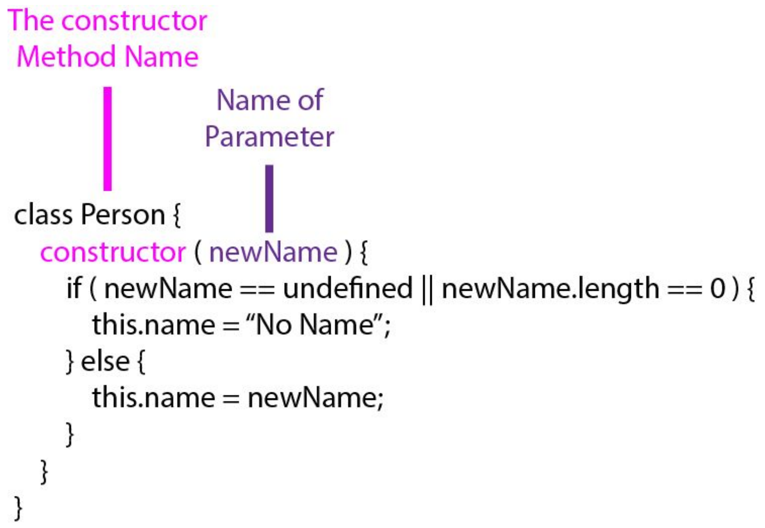
Note that the constructor method is not only used for accepting arguments during object creation, but it is also often used for object setup and default value assignments.

It gives us a place to organize and decide which properties need to be inside of objects built out of class. Let's try writing a constructor method to build on our Person class we were writing.

Constructor Method

Constructor methods help us define which properties and default values should be in objects instantiated from a specific class.

Inside of our class, we can add a constructor method. Note that “constructor” is a reserved method name, and is used exclusively for the purpose we have described.



```
class Person {
  constructor ( newName ) {
    if ( newName == undefined || newName.length == 0 ) {
      this.name = “No Name”;
    } else {
      this.name = newName;
    }
  }
}
```

The this Keyword

In classes, the this keyword let an object reference itself. This way, when writing a class, you can make use of a potential future object's properties and methods inside of itself.

With Person, as an example, we may create 10 people, but the constructor and any other values / functionality needs to be performed with any one of those objects. “this” is how we do that!

Our previous example features us assigning a value to the Person's name property…

this.name = “No Name”;

It is important that the this keyword works a bit differently depending on where you use it in your code… keep an eye out for other places and ways it can be used! We will cover at least one other in our time together.

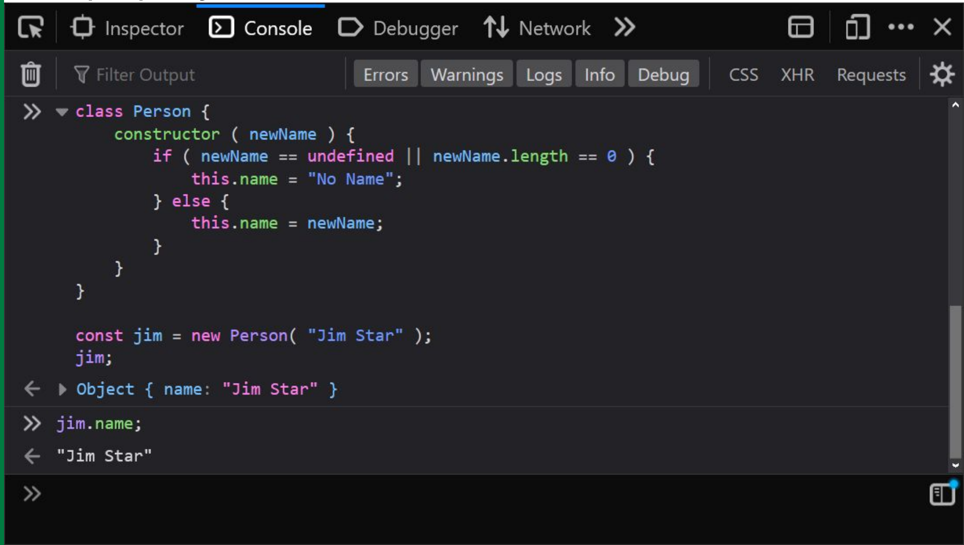
Using a Class; Accessing a Property

Let's try this Person class out, and access a property we've set up.

Write your person class; for now, we'll just accept a name.

Once we have the name value, we'll assign it as the object's "name" property value.

Remember, we can access a property by calling the object by name, following that with a period and the property's name.



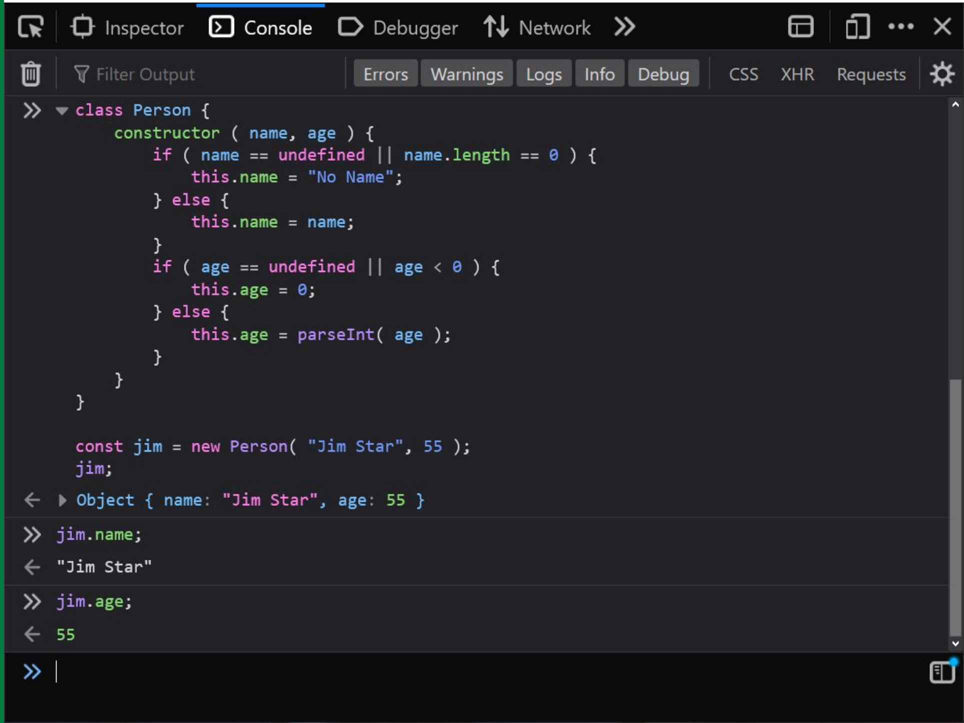
```
>> class Person {
       constructor ( newName ) {
           if ( newName == undefined || newName.length == 0 ) {
               this.name = "No Name";
           } else {
               this.name = newName;
           }
       }
   }

   const jim = new Person( "Jim Star" );
   jim;
<- Object { name: "Jim Star" }
>> jim.name;
<- "Jim Star"
>>
```

Let's Add Another Property

Before we dive into a basic method, let's add another property! This will give us more to play with.

So, our Person object has a name!

Now, how about an age… Add another parameter and couple of assignments in your constructor.

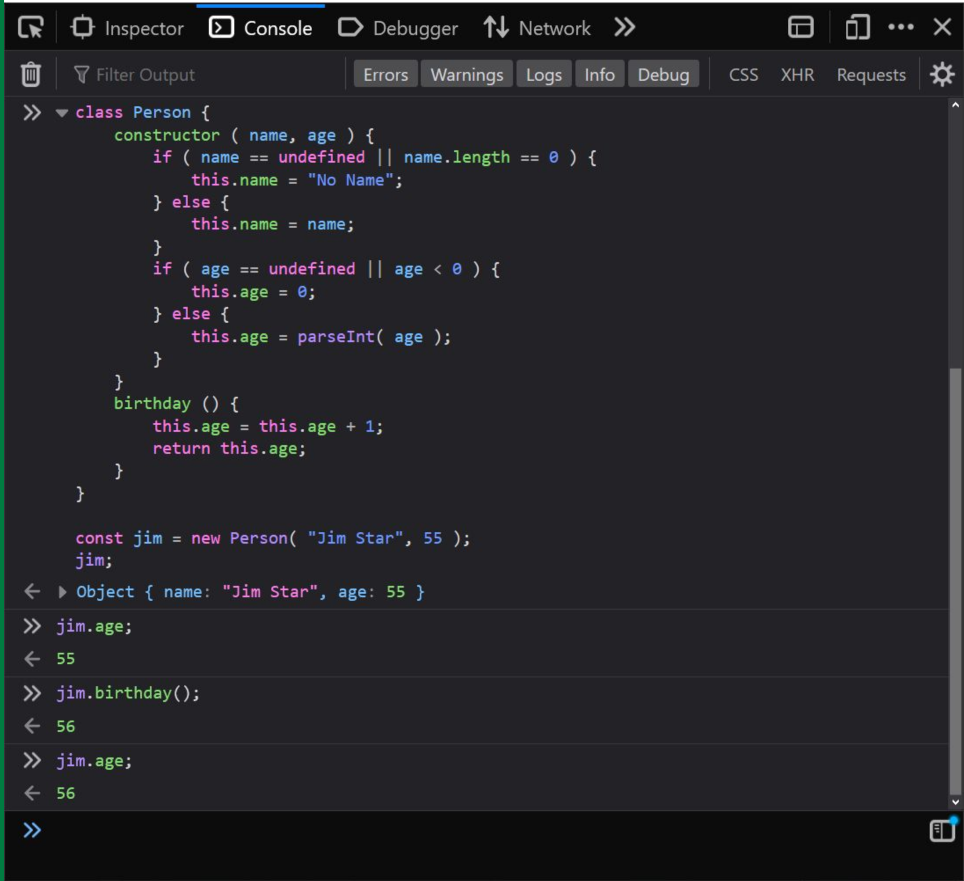


```
>> class Person {
       constructor ( name, age ) {
           if ( name == undefined || name.length == 0 ) {
               this.name = "No Name";
           } else {
               this.name = name;
           }
           if ( age == undefined || age < 0 ) {
               this.age = 0;
           } else {
               this.age = parseInt( age );
           }
       }
   }

   const jim = new Person( "Jim Star", 55 );
   jim;
<- ▶ Object { name: "Jim Star", age: 55 }
>> jim.name;
<- "Jim Star"
>> jim.age;
<- 55
>>
```

Methods

Constructor may be built in, but we can make our own methods. Remember: they're like functions, but stored in objects.

Add a birthday method and run it like so...

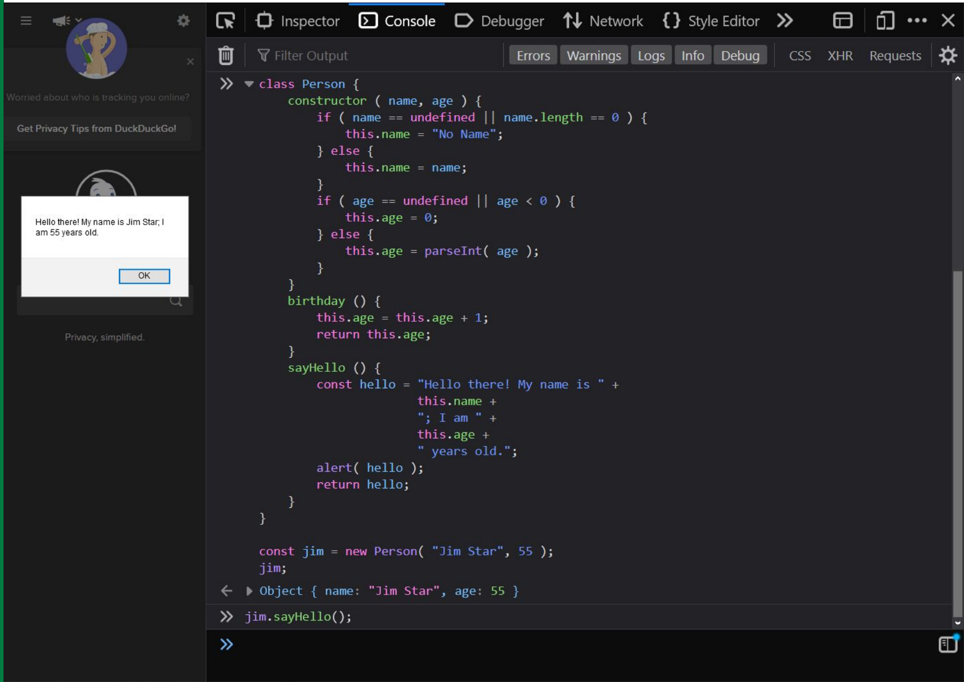


```
>> class Person {
       constructor ( name, age ) {
           if ( name == undefined || name.length == 0 ) {
               this.name = "No Name";
           } else {
               this.name = name;
           }
           if ( age == undefined || age < 0 ) {
               this.age = 0;
           } else {
               this.age = parseInt( age );
           }
       }
       birthday () {
           this.age = this.age + 1;
           return this.age;
       }
   }

   const jim = new Person( "Jim Star", 55 );
   jim;
<- Object { name: "Jim Star", age: 55 }
>> jim.age;
<- 55
>> jim.birthday();
<- 56
>> jim.age;
<- 56
>>
```

One Method More

Great! Our people can celebrate birthdays. How about we do one more method… Let's have them say hello!

We can use concatenation to access the Person's name and age and display a message string containing that information!



Worried about who is tracking you online?

Get Privacy Tips from DuckDuckGo!

Hello there! My name is Jim Star; I am 55 years old.

Privacy, simplified.

```
class Person {
    constructor ( name, age ) {
        if ( name == undefined || name.length == 0 ) {
            this.name = "No Name";
        } else {
            this.name = name;
        }
        if ( age == undefined || age < 0 ) {
            this.age = 0;
        } else {
            this.age = parseInt( age );
        }
    }
    birthday () {
        this.age = this.age + 1;
        return this.age;
    }
    sayHello () {
        const hello = "Hello there! My name is " +
                      this.name +
                      "; I am " +
                      this.age +
                      " years old.";
        alert( hello );
        return hello;
    }
}

const jim = new Person( "Jim Star", 55 );
jim;
← Object { name: "Jim Star", age: 55 }
» jim.sayHello();
```

What We’ve Covered so Far

With classes and objects we’ve gone over…

- Declaring a class
- Creating an instance of that class
- Assigning and accessing properties
- Declaring and executing methods