**JavaScript: Novice to Ninja**

**Chapter 8: Forms**

Tag <form></form>

Contain form controls: input fields, select menus and buttons

action attribute is the URL that the form will be submitted to

const form = document.form[0];

is the same as

const form = document.getElementsByTagname(‘form’)[0];

**Form Events**

focus occurs when an element is focused on.

blur occurs when the user moves the focus away from the form element

change occurs when the user moves the focus away from the form element after changing it

placeholder give a hint as to what should go in the input box

**Form Controls**

<input> text, check boxes, radio buttons, passwords, and file uploads.

<select> menus for drop-down lists.

<textarea> for longer text entries.

<button> for submitting and resetting form.

**New** input **Element Attributes**

autofocus gives focus to an element when a page is loaded

placeholder give a hint as to what should go in the input box

maxlength limits the number of characters that can be entered

**Text Input Fields**

Default type is text, it is not required to be explicitly defined

type=”password”, is used to enter passwords or secret information

type=”checkbox”, returns a true or false value. Can select multiple

type=”radio”, also returns a true or false value. Can only select one

type=”hidden”, these are not seen by the browser, but contain values that is submitted with the form.

type=”file”, used to upload files

**Other Input Fields**

type=”number”, has min & max options

type=”tel”,

type=”color”, pops up a box of optional colors

type=”email”, gives and error if there is no ‘@” in the input field

type=”date”, gives a calendar to select from

**Form Validation**

This is the process of checking if a user entered the information in the form correctly

\*Validation can be done front-side with JS, but should also be done on the back-side

You can use the built in feature to require a field be filled out like this:

<input type=”text” required>

You can also use JS to create custom validation.

\*Use an event listener to give instant feedback instead of waiting for the form to be submitted

Submit button can be disabled until validation is complete like this

<input type=”submit” disabled>

Once validation is done, you can enable submit like this:

document.getElementById('submit').disabled = false;

**JavaScript: Novice to Ninja**

**Chapter 12: Object Oriented Programming (aka OOP)**

This involves separating the code into objects that have properties and methods.

This approach keeps related pieces of code encapsulated in objects that maintain state throughout the life of the program

Objects can also be reused or easily modified, as required.

**Encapsulation**: The inner workings are kept hidden, only the essential functionalities are exposed

**Polymorphism**: The same process can be used for different objects. (Various objects can share the same method, but have the ability to override shared methods with more specific implementation.

**Inheritance**: We can take an object that already exists and inherit all its properties and methods, then we can improve on it by adding new properties and methods.

**Classes**: JavaScript uses a proto-type inheritance model

* **Constructor functions**

Parentheses are not required when instantiating a new object, as long as there is not a need for default arguments

\*JavaScript contains built-in constructor functions that can be used to crate objects, arrays and functions instead of literals

The easiest way to create an object is to use literal syntax:

const literalObject = {};

You could also use the object constructor:

const literalObject = new Object();

You can still use a literal also:

literalObject instanceof Object;

* **Using classes in JavaScript**

**Static Methods**

The static keyword is used in class declaration to create a static method.

\*A static method is called by the class directly and not by an instance of the class

\*Static methods are not available to instances of the class

* **Prototypes**

Prototyping allows you to add augment a class with extra methods and properties after it has been created.

This can be done like this

Turtle.prototype.attack = function(){

return `Feel the power of my ${this.weapon}!`;

}

**QUESTION:** If an object is instantiated and a new method is added to the prototype, does the already instantiated object have access to that method or is it only available to newly instantiated objects?

**ANSWER:** The prototype object is live, so if a new property or method is added to the prototype, any instances of its class will inherit the new properties and methods automatically, even if that instance has already been created.

To find the prototype of an object you can use Object.getPrototypeOf()

To find out if a property is inherited or is the objects own, you can use hasOwnProperty()

Prototype properties are shared by every instance of the class.

\* Never use arrays or objects as a default value in prototype.

* **Public and private methods**

This doesn’t seem too different than other stuff you have learned

* **Inheritance**

**Prototype Chain –** each prototype has its own prototype

**Enumerable Properties -** If they aren't enumerable, this means they will not show up when a for-in loop is used to loop through an object’s properties and methods.

There is a method called propertyIsEnumerable() that every object has (because it’s a method of Object.prototype ) that can be used to check if a property is enumerable

* **Creating objects from objects**

**Extends** - A class can inherit from another class using the extends keyword in a class declaration. This creates a child class.

Inside the child class, the word ***super*** refers to the parent class.

* **Adding methods to built-in objects**

This practice is known as ***monkey-patching***

The current consensus in the JS community is that this shouldn't be done, so you should avoid monkey-patching any of the built-in object constructor prototypes, unless you have a very good reason.

An alternative way to avoid causing problems is to use extends to subclass a built class

* **Mixins**

A mixin is a way of adding properties and methods of some objects to another object without using inheritance. It allows more complex objects to be created by ‘mixing’ basic objects together.

One way to think about the difference between prototypal inheritance and inheritance from mixin objects is to consider whether an object **is** something or whether it **has** something

* **Chaining functions**

If a method returns this , its methods can be chained together to form a sequence of method calls that are called one after the other.

A big drawback with this technique is that it can make code more difficult to debug. If an error is reported as occurring on a particular line, there is no way of knowing which method caused the error

* **This and that**

When functions are nested, this loses its scope and point to the global object inside a nested function.

A common solution is to set the variable that to equal this before the nested function, and refer to that in the nested function instead of this

bind() – is a method for all functions and is used to set the value of this in the function

Use for-of instead of forEach(), it doesn’t require a nested function

Arrow functions do not have their own this context so this remains bound to the original object

* **Borrowing methods from prototypes**

It’s possible to borrow methods from objects without having to inherit all their properties and methods.

You have to use the call() method to use a borrowed property.

**JavaScript: Novice to Ninja**

**Chapter 15: Modern JavaScript**

* **Libraries**
* **Modular JavaScript**
* **MVC frameworks**
* **Package managers**
* **Optimizing your code with minification**
* **Build processes using Webpack**
* **Our project ― we'll separate our code into a modular architecture and prepare for it for deployment.**