Chapter 1—Novice to Ninja, 2nd Edition.

Programming

--Low-leveling programming languages are machine code and assembly language.
--High-level programming languages is easy for humans to read and write…then it is complied into machine code and executed.
--Script Language is high-level but it is translated at machine code time.

JavaScript

--It is a versatile and easy program to use.

What is a dynamic language?

History of JavaScript

--JavaScript was invented to make webpages interactive. It has nothing to do with Java. Netscape was the one to develop it. Microsoft had their own versions which were Jscript and VBScript. It was easy to use and quickly became popular, even with people who didn't understand the big picture.

The Browser Wars

Netscape added proprietary features which meant that two versions had to be programmed. In the end Microsoft won and standards were started and vendors started working together.

Web 2.0…

Ajax and XML allowed pages to have relevant parts updated instead of reloading the whole page. JS became more powerful and flexible.

Standards

JS runs much faster now thanks to companies developing faster engines.

HTML5 is the dominant standard for web development.

Node.js- allows javascript to also run not on browers. It has lead to Isomorphic JS…it can be run on the client side or server side.

JavaScript Versions—JS is a superset of ECMAScript.. The latest version mostly used is ES6.

JS has a bright future. It is becoming more powerful and expressive. It is being used in Single page applications a lot.

Installation of Node…
Node package manager—the largest code repository in the world.
Babel—a transpiler that will convert code written in the latest version of JS into code that can be run in most browsers.

Install Node
All installed.

JavaScript in the Console--Don't understand REPL

Can use Browser Console—E66 Console

Text Editors—Some good free options…Atom, Sublime, Brackets
Integrated Development Environment (IDE)???

Online Options—CodePen, JSFiddle, JS Bin

JS works mostly in browsers

Three Layers of the Web-CSS. HTML. And JS—keeping layers separate is best practice.

Unobtrusive JS—keeping JS in a separate file and not written in HTML under <script>…but putting the file name under the <script>

Graceful degradation—programing so that the website works best in modern browsers but will still work reasonable in the other browsers.

Progressive enhancement…you build a page with the basics and add features as they become available in newer browsers.

Backward Compatibility—legacy code still needs to work the same way.

syntactic sugar—It allows code to be written in a simpler, easier way.

Transpilers convert code from one version to another.

Babel is a good transpiler to do this.

Chapter 2

Comments are good to help others understand. Use // and /* */ when writing comments.

Semi-colons are not necessary at the end of statements but are good practice to avoid problems elsewhere.

{ }. Go around blocks…no semi-colon after a block.

You can use as much whitespace as you want to make it look nice and easy to read.

Don't use reserved words that JS uses as code…and those that should be reserved.

Primitive Data Type…string, Boolean, symbol, number, undefined, null
If it is not a data type then it is an object—array, function, object literal. Use "typeof" to find out what the data type is.

"Variables are used in programming languages to refer to a value stored in memory."

Use "const" for variables that won't be reassigned. "Let" for variables that could be reassigned.

What about "var"?

```
const name = { value: 'Alexa' }; // an object
"value is the key or the
```

Const can not be changed and can be helpful in preventing errors.

Var doesn't have a scope but can be found still in older codes.

Scope has two types…local and global

Global scope is found outside and block and can be used anywhere…sometimes you want to use the same name in different code modules so be careful where you use this.

Local Scope…variables are created inside a block and can only be used in that block. Both let and const work here.

If a variable is used without let or const are global.

Naming
Use names that make sense and are easy to follow. Names can start with letters, underscore, or $. Although underscore and $ are used to name special variables and better not used.

Case sensitive, camel case is best, underscores can be used between words, but camel case is best.

Direct Assignment—you directly change the value of to a primitive variable.

Non-primitive variables are assigned by reference. When two variables are referencing each other, than when one changes, the other changes.

Strings are characters, symbols, letters with quotes around them.

No numbers with quotes make strings?

String literal notation

Backslashes can be used before an apostrophe before to make and ‘ in a string

String Property and Methods.

Properties are information about a string. And methods are functions within an object or an action performed on a string.

Properties are accessed wth dot- notation…like .length, you can also use brackets…
name['length'];

Methods are actions you can perform on primitive data and objects.

Lots of methods available…they use () at the end of the method name.

Template Literals

--the use of the backtick let’s strings be literal with the use of ${ } to insert JS. /n are line breaks

Symbol() can create symbols…example “const uniqueID = Symbol('this is a unique ID');”

For() can assign shared symbols to two variables

Numbers can be integers of pointing float (decimal) numbers

JS supports Hexadecimal, Octal, and binary numbers

1e6 means multiplied by the power of 10 six times

Numbers have numbers but should be used carefully. .toFixed ()

Operators. + - / * % (for remainder) **(for exponentiations)
Changing the values of variables using operators.

Incrementing Values-- ++. Use it before to increase before returned value or after for increasing after returned value.

"—" Works the same way

"Infinity" means the number is too big for JS to handle

NaN is an error message which means "not a number"

Type Coresion is when JS will force a string number to a number value to make the formula work.

Number() converts strings to numbers…when it's a number string or convert numbers to strings…string() method

parseInt converts a string numbers…parseInt(*number, base*)…it removes numbers after the decimal. Use parseFloat() to keep the numbers after the decimal.

Undefined…variables that haven't been assigned a value.

Null means no value…placeholder

"null is coerced to be 0 , making the sum possible whereas undefined is coerced to NaN , making the sum impossible to perform."

Boolean…true and false. Some values are always false such as 0 or an empty string…undefined, null, NaN

Logical Operators.

! = means "NOT". So !True is actually false.
!! = brings back the Boolean
&& = AND
|| = OR

"Bitwise operators work with operands that are 32-bit integers. These are numbers written in binary (base two) that have 32 digits made up of just 0 s and 1 s."

Do people really use bitwise operators?

Bitwise AND, Bitwise NOT, Bitwise OR, Bitwise XOR,
Bitwise shift operators <<. >>

Examples:

```
3 << 1; // multiply by 2
<< 6
16 >> 1; // divide by 2
<< 8
5 << 3; multiply by 2 cubed (8)
<< 40
```

Comparisons

Equal
--Soft equal == it compares but is soft on data type and sometimes coerce values to be the same.
--Hard equal=== true including data type.
Always use hard equality when testing.
Use with the alphabet too in order

Inequality
--Soft !=
--Hard !==

Greater than and less than
<. >
Greater than or equal to. Or less than or equal to…they are only soft.
<=. >=

For hard equalities use an OR and === to get the answer

Chapter 3

An array is an ordered list of values

Const = arrayName [ ] To create an array or const = new Array()---the first one “array literal” uses less typing so is better. They are empty right now…undefined

It is an object—for typeof

Const = heroes[];
Heroes[0] = “superman”;---assign like variables using the index numbers to any position.
Skipped index number are undefined.

Arrays can have multiple types of data

--delete heroes[i] to delete an object…which then becomes undefined.

Arrays can be deconstructed. Each item in the array has it's own value too.

Array Properties and Methods
--.length can be used to find the last item in a list. heroes[heros.length-1]

--pop() removes the last item from an array
--shift() removes the first item from an array
--push() adds an item to the end
--unshift() adds an item to the beginning
--concat() to merge arrays
--use spread operator …. Similarly

--join() joins to arrays and makes them a string.
--splice() removes and replaces.
--reverse(). Reverses the order fo the array
--sort(). Sorts the array alphabetically permentally…numbers are sorted by their first digit.
--indexOf(). Returns the index number of an item or a "-1" if it is not there.
--includes() if array contains, it returns true

Multidimensional Arrays—an array of arrays

Example "const coordinates = [[1,3],[4,2]];"
to find a value in it…coordinates [0] [0]…. Indicates which array and the second which value in the array….that one is the first value of the first array.

Flatten the array or the nesting using the spread operator …

Sets are collections of unique values…no duplicate—track data without worry about duplicates.

CREATING SETS
Use new Set()

Const list = new Set();

Add items into a set with the add() method.

--list.add(1);

-- const numbers = new Set([1,2,3]);--set up and added at once
Repeated values only appear once.----new Set ('hello'). Would only have one "l", separate words all need to use the .add() method

Non-primitive values are all unique. ---there is no type coercion

They are not assigned index numbers

SET METHODS
--size()
--has()---is the value the set…returns true or false (more efficient in sets than includes())
--delete()…removes an item from the set
--clear()removes all values from the set
--Array.from() to convert from set to array
--new Set() to convert an array into a set ex..
You can covert it into an array by using the spread operator ---const shoppingArray = [...shoppingSet]

```
const duplicate = [3, 1, 4, 1, 5, 9, 2, 6 ,5,3,5,9];
<< [ 3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 9 ]
const nonDuplicate = [...new Set(repeatedArray)];
<< [ 3, 1, 4, 5, 9, 2, 6 ]
```

???? Wouldn't it be "duplicate" instead of "repeadedArray"

Weak sets avoid this situation by garbage collecting any references to a 'dead object' that's had its original reference removed.

New WeakSet()—then use add()—use has() to confirm if something is in it.

MAPS

Maps are another way of keeping a list of key and value pairs.
--you can use any type of data for the key.
--you can use the "size" property to find out how many keys-pairs there are.
--maps focus on storage and retrieval
--use get() to retrieve values

Creating Maps const newMap = new Map();
--use set() to add items….newMap.set(key, pair);
-- the => is used to show the key to pair relationship…key=>pair
--has() to see if the a map has a key
--delete()—to remove a key or value
--clear()—removes all keys and pairs
--Array.from() to convert to an array

WEAK MAPS

Weak maps can use the has() , get() , set() and delete() methods in the same way as a regular map…for optimizing memory leaks

LOGIC

-If statements

--if (condition) {}

--if (condition) {}
Else {}

Shorthand for if/else is ---condition ? true action : else action;
Example…console.log(`n is a ${(n%2 === 0)? 'even' : 'odd'} number`);---more succinct but harder to read.

Switch---when there are lots of conditions in an if/else if/else statement, switch can be easier to use.

--switch (condition) {
case
case
case
default (used if no conditions are met)
break (ends)
}

LOOPS

<u>While Loop</u>
--while (condition) {
Do something
}

Beware of Infinite Loops

<u>Do…While Loops</u>
--do {
Something
} while (condition)

<u>For Loops</u>

--for (initialization; condition; after) {do something}

Nest Loops

```
--for(let i=1; j<13; i++){
--for(let i=1; j<13; i++){
        console.log(`${j} multiplied by ${i} is ${i*j}`);
```
--Isn't the second loop suppose to be "j=1"?

Looping over Arrays

```
For (let i=0; x=array.length; i<x; i++) {
Do something
}
```

Array indices start at 0, so the counter has to start as 0

Looping Over Sets
--sets are enumerable, they loop over each value in the set…except weak sets are non-enumerable

Ex. Const letters = new Set ('hello');

```
--for (const letter of letters) {
--console.log(letter);
}
```

<<h e l o

Looping Over Maps

Maps are enumerable—loops iterate over the maps in the same order that they were added.
--key()—a method to iterate over each key with the for-of loop.
--values()—a method to iterate over each value (pair) similar to the key() method.
--entries()—lets you access both.

for...of
The **for...of statement** creates a loop iterating over iterable objects, including: built-in String, Array, array-like objects (e.g., arguments or NodeList), TypedArray, Map, Set, and user-defined iterables. It invokes a custom iteration hook with statements to be executed for the value of each distinct property of the object.

Example:

```
const array1 = ['a', 'b', 'c'];
for (const element of array1) {
  console.log(element);
}

// expected output: "a"
// expected output: "b"
// expected output: "c"
```

From -- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of

The **conditional (ternary) operator** is the only JavaScript operator that takes three operands: a condition followed by a question mark (?), then an expression to execute if the condition is truthy followed by a colon (:), and finally the expression to execute if the condition is falsy. This operator is frequently used as a shortcut for the if statement.

Condition ? *something to execute if true* : *something to execute if false*;

https://aleshana.github.io/WDD-330-Portfolio/index.html

Chapter 4---

Function declaration

```
function hello(){
    console.log('Hello World!');
}
```

Evoking…

```
hello();
<< 'Hello world!'
```

Function Expression

```
const goodbye = function bye(){
    console.log('Goodbye World!');
```

```
};

//to envoke
goodbye();
<< 'Goodbye World!'
```

Without the () it just points to the function.

Return Values

Parameters—the parameters are set when the function is defined.
Arguments—when the function is invoked the arguments are provided
If a parameter is not provided as an argument when the function is invoked, the function will still be invoked, but the parameter will be given a value of `undefined`

Functions don't create arrays, but objects with lists.
Use "rest"—use the spreader opertator after an array name to create an array.

```
function rest(...args){
    return args;
}
```

---for in loops through property names (**key**)---for (let = variableName in objectName) {statements}---innumberable objcets but not iterable—will return **index number for arrays and user defined properties(**the key)---to call each value use variableName[objectName]

--for of loops through property **values**. for (variableName of Object) {statements}---for interable objects—passes the **values** of the array.

Default Parameters

```
function hello(name='World') {
    console.log(`Hello ${name}!);
}
```

Override by just adding a regular argument.
Default parameters need to come after the non-default parameters.

Arrow Function

```
const square = x => x*x;
```

no "return" required. No () on single parameters, they don't bind "this"

```
--const add = (x,y) => x + y;
```

For no parameters

```
const hello = () => alert('Hello World!');
```

longer functions need {}...best for short functions.

Function Hoisting—all variables and function declarations are at the top of the current scope.

Variable Hoisting
"var" variables and function expressions are not hoisted…and can't be invoked until after it appears in the code.

CALLBACKS
Functions can be given as a parameter to another function. The callback is provided as a parameter and then invoked in the first function.

Just pass the argument of the function…no (), to invoke the function in the first function then use ()

```
sing('Let It Go',()=>{ console.log("I'm standing on my head.");});
<< 'I'm singing along to Let It Go.'
'I'm standing on my head.'
```

Only good for anonymous short functions…not usually recommended.

For Sorting with numbers not strings..use the sort() method with the following function

```
function numerically(a,b){
    return a-b;
}
```

Then

```
> [1,3,12,5,23,18,7].sort(numerically);
<< [1, 3, 5, 7, 12, 18, 23]
```

--forEach

--arrayName.forEach((refersToValueInArray, currentIndex)
Dosomething);

--map()—like the forEach it interates over the array but then makes a new array with the newArray

```
[1,2,3].map( x => 2 * x);
<< [2,4,6]
```

Or

```
[1,2,3].map( square )
<< [1, 4, 9]
```

--x is the parameter which refers to the items in the array

--reduce()—combines each result to return a single value (sum)—two parameters=1st is representsSum, 2nd is currentValue.

--reduce()
--filter()

Chaining methods together.

Chaining example

```
[1,2,3].map( x => x*x ).reduce((acc,x) => acc + x );
<< 14
```

More notes