This section of Notes in for chapter 5
There was also reading from an external link whose notes can be found first

Object Methods, "this"

Objects:
- Created to represent entities of real world
- users, orders, etc

A function that is a property of an object is called a **method**

```
let user = {
  // ...
};

// first, declare
function sayHi() {
  alert("Hello!");
};

// then add as a method
user.sayHi = sayHi;

user.sayHi(); // Hello!
```

method sayHi of the object .user

"this" in methods

To access an object, a method can use this keyword
The value of this is the object "before dot", the one used to call the method

```
let user = { name: "John" };
let admin = { name: "Admin" };

function sayHi() {
  alert( this.name );
}

// use the same function in two objects
user.f = sayHi;
admin.f = sayHi;

// these calls have different this
// "this" inside the function is the object "before the dot"
user.f(); // John  (this == user)
admin.f(); // Admin  (this == admin)

admin['f'](); // Admin (dot or square brackets access the method
```

The rule is simple:
if obj.f() is called, then this is obj during the call of f.
So its either user or admin here

Arrow functions have no "this"

If we reference this from such a function, its taken from the outer "normal function"

```
let user = {
  firstName: "Ilya",
  sayHi() {
    let arrow = () => alert(this.firstName);
    arrow();
  }
};

user.sayHi(); // Ilya
```

arrow() uses this from the outer .sayHi() method

Summary

- Functions that are stored in object properties are called "methods".
- Methods allow objects to "act" like object.doSomething().
- Methods can reference the object as this.
- The value of this is defined at run-time.
-
- When a function is declared, it may use this, but that this has no value until the function is called.
- A function can be copied between objects.
- When a function is called in the "method" syntax: object.method(), the value of this during the call is object.

Chapter 5 : Objects

Arrays & functions are both objects

Object:

- a self contained set of related values and functions
- act as a collection of named properties that map to JS values, (strings numbers, bools, etc.
- If a property's value is a function, its known as a method
- often used to keep related info & functionality together

A Super Example

Object literal - object that is created directly in the language by wrapping all properties in curly braces { }

- allow objects to be created quickly w/out the need for defining a class

example of object literal that describes Man of Steel:

```
"const superman = {
name: 'Superman',
'real name': 'Clark Kent',
height: 75,
weight: 235,
hero: true,
villain: false,
allies: ['Batman','Supergirl','Superboy'],
fly() {
  return 'Up, up and away!';
}
};"
```

- all objects are mutable @ any time when a program is running

Creating Objects

create an object literal:
→ enter a pair of curly braces
→ use a constructor function *this method is not recommended

Accessing Properties

access properties of an object by:
- using the dot notation → `superman.name` << 'Superman'
- using bracket notation → a string inside square brackets

```
superman['name']
<< 'Superman'
```

Computed Properties

the ability to create objects w/ computed property keys
meaning: JS code can be placed in [] & the property key will be the return value.

- Symbol data type can also be used as a computed property key
- access property using [] notation
- add new property to object using symbol as key if [] notation is used

Calling Methods

- to call object's method, use dot or [] notation
- calling method is the same as invoking a function

```
Superman.fly()
<< 'Up, up and away!'
```

```
superman['fly']()
<< 'Up, up and away!'
```

Checking if Properties or Methods Exist

in operator can check whether an object has a particular property

```
'City' in superman
<< false
```

hasOwnProperty() can check whether an object has a property that is its own, rather than one that has been inherited from another object

```
superman.hasOwnProperty('city');
<<false
superman.hasOwnProperty('name');
<<true
```

this method will ONLY return any properties that belong to that particular object

Finding all the Properties of an Object

we can loop thru all of an object's properties & methods by using a for in loop to log all the properties of the superman object:

```
for(const key in superman) {
console.log(key + ": " + superman[key]);
}
<< "name: Superman"
<< "real name: Clark Kent"
<< "height: 75"
<< "weight: 235"
<< "hero: true"
<< "villain: false"
<< "allies: Batman,Supergirl,Superboy"
<< "fly: function (){
  console.log(\"Up, up and away!\");
}"
```

we create a variable called key, then iterate over the properties of the superman object & use key to log the property name & superman[key] to look up the value of each property

key-value pairs are returned in arrays, but they can be destructed & accessed individually

Adding Properties

- done by assigning a value to the new property

* properties don't always appear in the order they were entered

add a new city property

```
superman.city = 'Metropolis';
<< 'Metropolis'
```

Now if we take a look at the superman object, we can see that it has a city property:

```
superman
<< { name: 'Superman',
   'real name': 'Clark Kent',
   height: 75,
   weight: 235,
   hero: true,
   villain: false,
   allies: [ 'Batman', 'Supergirl', 'Superboy' ],
   fly: [Function: fly]
   city: 'Metropolis' }
```

Changing Properties

change the value of objects properties using assigning

```
“superman['real name'] = 'Kal-El';
<< 'Kal-El”
```

Removing Properties

delete operator can remove any property

```
delete superman.fly
<< true
```


```
.superman
<< {"allies": ['Batman', 'Supergirl', 'Superboy'], 'city': 'Superman', 'real name': 'Kal-El', 'villain': false,
'weight': 235}
```

Nested Objects

Objects can contain other objects

```
const jla = {
superman: { realName: 'Clark Kent' },
batman: { realName: 'Bruce Wayne' },
wonderWoman: { realName: 'Diana Prince" },
flash: { realName: 'Barry Allen' },
aquaman: { realName: 'Arthur Curry' },
}
```

access nested objects by referencing each property name:

```
jla.wonderWoman.realName
<< "Diana Prince"

jla['flash']['realName']
<< "Barry Allen"
```

Objects are copied by Reference

if a variable is assigned to an object that already exists, it will simply point to the exact same space in memory

Objects as Parameters to Functions

"An object literal can be passed as a parameter to a function. This is useful when there are a large number of parameters, as it allows the arguments to be provided by name and in any order"
"you don't have to remember the order to enter them when invoking a function"

this

the keyword this refers to the object that it is within

```
const dice = {
sides: 6,
roll() {
  return Math.floor(this.sides * Math.random()) + 1;
}
}
```

This object has a sides property and a roll() method. Inside the roll() method we use this.sides to refer to the value of the object's sides property

Namespacing

Naming collisions = when the same variable or function name is used for different purposes by code sharing the same scope
To solve this problem, use the object literal pattern to create a namespace for groups of related functions

Built-in Objects

JSON - used to exchange info between web services
- sweet spot between both human- and machine- readable
- a string representation of the object literal notation
- however, differences include:
  1. property names must be double-quoted
  2. permitted values are double-quoted strings, numbers, true, false, null, arrays, & objects
  3. Functions are not permitted values

parse() takes a string of data in JSON format & returns a JS object
stringify() takes a JS object & returns a string of JSON data

The Math Object

- built in object that has several properties representing mathematical constants
- all properties & methods of **math** object are immutable & unchanging

Mathematical Constants

- **Math** has 8 properties that represent a mix of common math constants

```
Math.PI // The ratio of the circumference and diameter of a circle
<< 3.141592653589793

Math.SQRT2 // The square root of 2
<< 1.4142135623730951

Math.SQRT1_2 // The reciprocal of the square root of 2
<< 0.7071067811865476

Math.E // Euler's constant
<< 2.718281828459045

Math.LN2 // The natural logarithm of 2
<< 0.6931471805599453

Math.LN10 // The natural logarithm of 10
<< 2.302585092994046

Math.LOG2E // Log base 2 of Euler's constant
<< 1.4426950408889634

Math.LOG10E // Log base 10 of Euler's constant
<< 0.4342944819032518
```

Mathematical Methods

Math has methods to carry out variety of useful mathematical operations

Absolute Values

Math.abs()

Rounding Methods

Math.ceil() // rounds up
Math.floor() // rounds down
Math.round() // rounds to nearest integer
Math.trunc() // returns integer part of #

Powers & Roots

Math.exp() // raise # to the power of Euler's constant
Math.pow() // raise any # (1st arg) to the power of another # (2nd arg)
Math.sqrt() // returns positive sqr root of a #
Math.cbrt() // returns cube root of #
Math.hypot() // returns sqrt of the sum of the squares of all its arguments

Logarithmic Methods

Math.log() // returns natural logarithm of #
Math.log(Math.E) // Natural logs have a base of Euler's constant
Math.log2() // logarithms in base 2
Math.log10() // logs in base 10

Maximum & Minimum Methods

Math.max() // returns max number from arguments
Math.min() // returns min number from arguments

Trigonometric Functions

* all angles are measured in radians for these functions

Radians

- standard unit of angular measurement, equal to the angle of the circle's center corresponding to the arc that subtends it.

Rounding Errors

Computers have trouble dealing w/ decimal fractions & answers can vary platform to platform

Math.sin() // returns sine of an angle
Math.cos() // returns cosine of angle
Math.tan() // returns tangent of an angle
Math.asin() // returns the arcsine of number, results in angle
Math.acos() // returns the arccosine of number, results in angle
Math.atan() // returns the arctangent of number, results in angle

hyperbolic functions & their inverses

Math.sinh()	Math.asinh()
Math.cosh()	Math.acosh()
Math.tanh()	Math.atanh()

Random Numbers

```
Math.random()
6 * Math.random()
<< 4.5809...
Math.floor(6 * Math.random());
<< 4
```

The Date Object

Contains info about dates & times. Each object represents a single moment in time.

Constructor Function

A constructor function is used to create a new date object using the new operator:
const today = new Date()"

"To see what the date is, we use the toString() method"

"It's possible to create Date objects for any date by supplying it as an argument to the constructor function"
"the string passed to the Date constructor can be in a variety of formats"

The Epoch - 1st January 1970

"It results in a very large number and there is a potential problem looming in 2038 when the number of seconds since the Epoch will be greater than 2,147,483,647, which is the maximum value that many computers can deal with as a signed 32-bit integer"

"this problem will not affect JavaScript dates because it uses floating-point numbers rather than integers"

Getter Methods

"getter methods, which return information about the date object, such as the month and year"
"The getTime(), getTimezoneOffset() and getYear() methods don't have UTC equivalents"

UTC

"UTC is the primary time standard by which the world regulates clocks"

"The getDay() and getUTCDay() methods are used to find the day of the week that the date object falls on. It returns a number, starting at 0 for Sunday, up to 6 for Saturday"

"getDate() and getUTCDate() methods return the day of the month for the date object"

"getMonth() and getUTCMonth() methods can be used to find the month of the date object"

"getFullYear() and getUTCFullYear() methods return the year of the date object"

"There are also getHours(), getUTCHours(), getMinutes(), getUTCMinutes(), getSeconds(), getUTCSeconds, getMilliseconds(), and getUTCMilliseconds() methods that will return the hours, minutes, seconds and milliseconds since midnight"

"The getTime() method returns a timestamp representing the number of milliseconds since the Epoch"

"The getTimezoneOffset() method returns the difference, in minutes, between the local time on the computer and UTC"

Setter Methods

these methods can be used to change the value of the date held in a Date object

The RegExp Object

- regular expression (RegExp or regex)
- pattern that can be used to search strings for matches to the pattern
- common use is 'find & replace' type operations

Creating RegEXP

2 ways to create a regexp

1. (preferred) use the literal notation of writing the regexp between forward slashes that we've seen before:

```
const pattern = /[a-zA-Z]+ing$/;
```

"Using literal regular expressions takes less typing, but there are advantages to using the constructor function as it lets you create regular expressions using strings, which can be useful when the regular expression is provided from user input; in a form, for example."

RegEXP Methods

"The exec() method works in the same way as the test() method, but instead of returning true or false, it returns an array containing the first match found, or null if there aren't any matches"

Basic Regular EXPressions

"At the most basic level, a regular expression will just be a string of characters, so the following will match the string 'JavaScript':

```
const pattern = /JavaScript/;
<< /JavaScript/
```

Character Groups

"Groups of characters can be placed together inside square brackets. This character group represents any one of the characters inside the brackets"
"sequence of characters can also be represented by placing a dash [-] between the first and last characters"

"a \^ character is placed at the start of the sequence of characters with the brackets, it negates the sequence"

"These groups can be combined with letters to make a more complex pattern. For example, the following regular expression represents the letter J (lowercase or capital) followed by a vowel, followed by a lowercase v, followed by a vowel:

```
pattern = /[Jj][aeiou]v[aeiou]/;
<< /[Jj][aeiou]v[aeiou]/

pattern.test('JavaScript');
<< true
```

```
pattern.test('jive');
<< true

pattern.test('hello');
<< false”
```

Regular Expression Properties

“Regular expressions are objects, and have the following properties:

The global property makes the pattern return all matches. By default, the pattern only looks for the first occurrence of a match.

The ignoreCase property makes the pattern case-insensitive. By default, they are case sensitive.

The multiline property makes the pattern multiline. By default, a pattern will stop at the end of a line.

The following flags can be placed after a regular expression literal to change the default properties:

- g sets the global property to true
- i sets the ignoreCase property to true
- m sets the multiline property to true”

Special characters

“In a regular expression, there are a number of characters that have a special meaning, commonly known as metacharacters:

- . matches any character, except line breaks
- \w matches any word character, and is equivalent to [A-Za-z0-9_]
- \W matches any non-word character, and is equivalent to [\^A-Za-z0-9_]
- \d matches any digit character, and is equivalent to [0-9]
- \D matches any non-digit character, and is equivalent to [^0-9]
- \s matches any whitespace character, and is equivalent to [\t\r\n\f]
- \S matches any non-whitespace character, and is equivalent to [^ \t\r\n\f]”

Modifiers

“Modifiers can be placed after a token to deal with multiple occurrences of that token:

- ? makes the preceding token in the regular expression optional
- * matches one or more occurrences of the preceding token
- + matches one or more occurrences of the preceding token
- {n} matches n occurrences of the preceding token”
- {n,} matches at least n occurrences of the pattern
- {,m}matches at most m occurrences of the preceding token
- {n,m} matches at least n and at most m occurrences of the preceding token
- ^ marks the position immediately before the first character in the string
- $ marks the position immediately after the last character in the string

special characters or modifiers can be escaped using a backslash

/\?/ if you wanted to match a question mark

Greedy & Lazy Modifiers

- greedy modifiers will match the longest possible string
- adding an extra '?' after the modifier will turn it into a lazy modifier

A Practical Example

- match() method returns an array of all the matches
- by default, only the first is returned
- use the g flag to return all the matches

```
JavaScript'.match(/[aeiou]/g); // return an array of all the vowels
    << ['a', 'a', 'i']
```

Matched Groups

- Sub patterns can be created inside regexp by placing them inside parenthesis → these are known as capturing groups

Chapter Summary

- Objects are a collection of key-value pairs placed inside curly braces {}.
- Objects have properties that can be any JavaScript value. If it's a function, it's known as a method.
- An object's properties and methods can be accessed using either dot notation or square bracket notation.
- Objects are mutable, which means their properties and methods can be changed or removed.
- Objects can be used as parameters to functions, which allows arguments to be entered in any order, or omitted.
- Nested objects can be created by placing objects inside objects.
- JSON is a portable data format that uses JavaScript object literals to exchange information.
- The Math object gives access to a number of mathematical constants."
- The Math object can be used to perform mathematical calculations.
- The Date object can be used to create date objects.
- Once you've created a Date object, you can use the getter methods to access information about that

date.

- Once you've created a Date object, setter methods can be used to change information about that date.
- The Regex object can be used to create regular expressions.