Chapter 11 - Practice Set

1. Create a JavaScript class to create a complex number. Create a constructor to set the real and the complex part

2. Write a method to add two complex numbers in the above class

3. Create a class Student from a class Human. Override a method & see changes

4. See if Student is an instance of Human using instanceof keyword.

5. Use getters & setters to set and get the real and imaginary parts of the complex number

Chapter 11 - Object Oriented Programming

In programming we often take something and then extend it. for example we might want to create a user object and "Admin" and "guest" will be slightly modified variants of it

[[Prototype]]

JavaScript objects have a special property called prototype that is either null or references another object

```
┌─────────────────┐
│ prototype object │
└─────────────────┘
         ▲
         │  [[ Prototype ]]
┌─────────────────┐
│ Object │
└─────────────────┘
```
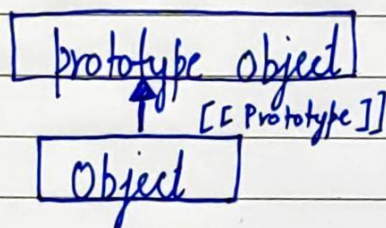
When we try to read a property from a prototype and its missing, JavaScript automatically takes it from the prototype. This is called "prototypal inheritance"

Setting Prototype

We can set prototype by setting __proto__
Now if we read a property from the object which is not in object and is present in the prototype, JavaScript will take it from prototype

If we have a method in object, it will be called from the object. If its missing in object and present in prototype, its called from the prototype

Classes and Objects

In object-oriented programming, a class is an extensible program-code template for creating objects, providing initial values for state (member variables) and implementation of behavior (member functions)

The basic syntax for writing a class is :

```
Class MyClass {
  // class methods
  constructor () { ... }
  method 1 () { ... }
  method 2 () { ... }
}
```

We can then use new MyClass () to create a new object with all the listed methods

The Constructor method

The Constructor () method is called automatically by new, so we can initialize the object there

Quick Quiz : Create a class user and create a few methods along with a constructor.

Class Inheritance

Class Inheritance is a way for one class to extend another class. This is done by using the extends keyword.

The extends keyword
extends keyword is used to extend another class.

        class Child extends Parent

We can create a class Monkey that inherts from Animal

```
class Monkey extends Animal {
    hide () {
        alert (`${ this.name } hides !`);
    }
}
```

```
let monkey = new Monkey ("Monu")
    monkey.run (7);   // from Animal
    monkey.hide ();
```

Method Overriding
If we create our own implementation of run, it will not be taken from the Animal class. This is called Method Overriding

super keyword
When we override a method, we dont want the method of the previous class to go in vain. We can execute it using super keyword.

        super (a, b)   →   call parent constructor

```
run () {
    super. run ()
    this. hide ()
}
```

Overriding Constructor

With a constructor, things are a bit tricky / different
According to the specification, if a class extends
another class and has no constructor, then the
following empty constructor is generated

```
class Monkey extends Animal {
    // auto generated
    constructor (... args) {
        super (... args);
    }
}
```

⟹ Happens if we dont write
     our own constructor

Constructors in inheriting classes must call super (...)
and do it before using this.

We can also use super.method() in a child method
to call Parent Method

Static method

Static methods are used to implement functions that
belong to a class as a whole and not to
any particular object

We can assign a static method as follows:

```
Class Employee {
    static sMethod () {
        alert ("Hey");
    }
}

Employee. sMethod ()
```

Static methods arent available for individual objects

Getters and Setters

Classes may include getters and setters to get &
set the computed properties

Example :

```
Class Person {
    ...

    get (name) {
        return this. _name;
    }

    set name (newName) {
        this. _name = newName;
    }
}
```

First the name property is changed to _name to
avoid the name collision with the getter & setter.
Then the getter uses the get keyword as shown above

Instanceof Operator

The instanceof operator allows to check whether an object belongs to a certain class

The syntax is :

< obj > instanceof < class >

It returns true if obj belongs to the Class or any other class inheriting from it