Session-03

Object Oriented Javascript

Thanos is on a mission to make his website standout from his rest of universe with Javascript

Web Fundamentals

Session-03

Object Oriented Javascript

Agenda : Javascript OOPS

01 Type Checking

02 All About Objects

03 Prototypes

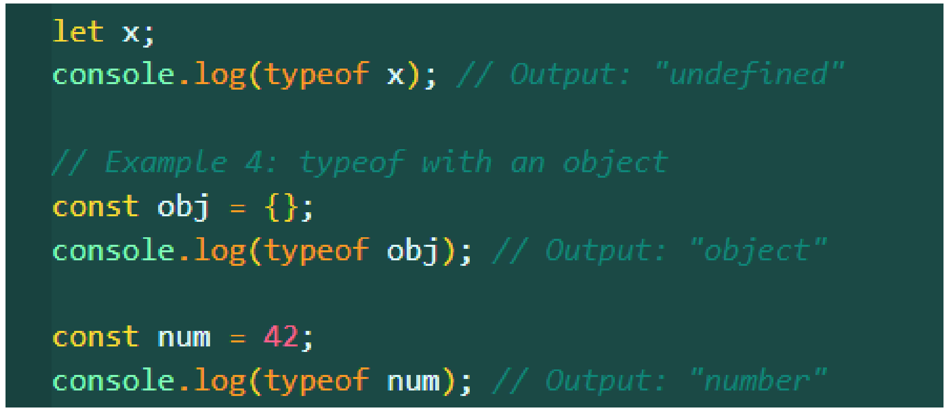
04 OOPS using ES5

05 OOPS using ES6

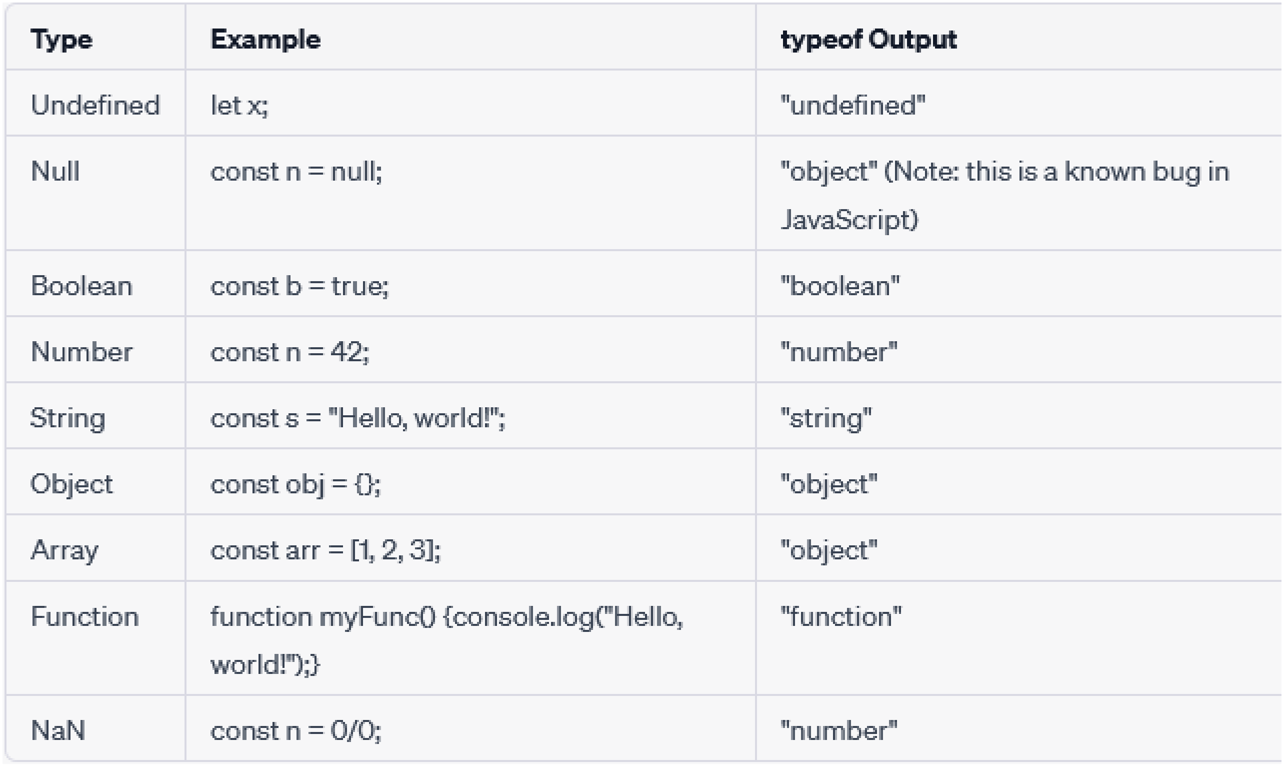
06 Hands-On

How **type checking** works in JavaScript

<TypeOf/> Operator



```
let x;
console.log(typeof x); // Output: "undefined"

// Example 4: typeof with an object
const obj = {};
console.log(typeof obj); // Output: "object"

const num = 42;
console.log(typeof num); // Output: "number"
```



Type	Example	typeof Output
Undefined	let x;	"undefined"
Null	const n = null;	"object" (Note: this is a known bug in JavaScript)
Boolean	const b = true;	"boolean"
Number	const n = 42;	"number"
String	const s = "Hello, world!";	"string"
Object	const obj = {};	"object"
Array	const arr = [1, 2, 3];	"object"
Function	function myFunc() {console.log("Hello, world!");}	"function"
NaN	const n = 0/0;	"number"

JS-Objects: Background

- JavaScript was **not originally designed as an object-oriented language.**
- JavaScript **was a Scripting language to** manage web applications
- ECMAScript 2015 (ES6) standard in 2015 introduced even more features to support object-oriented programming

Today, JavaScript is a fully functional object-oriented language that can be used to create complex and scalable applications.

JS-Objects: Background

Everything in JavaScript is Object except primitive types, definitions

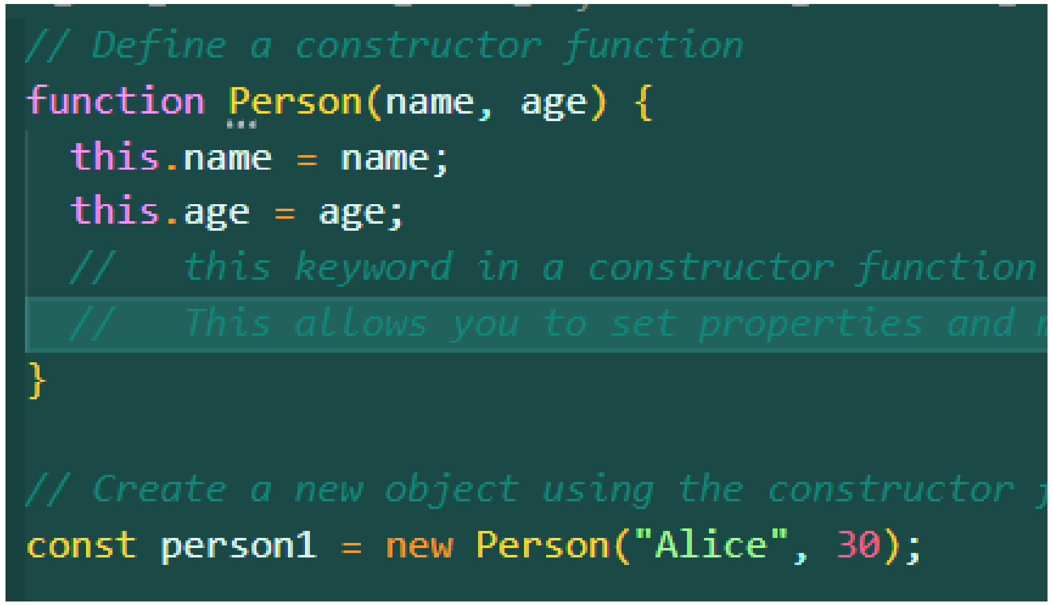
Let's learn all about Object

How to create Object

- Object.Create()
- Object literals / Object instantiation
- Using “new” keyword

Constructor Function

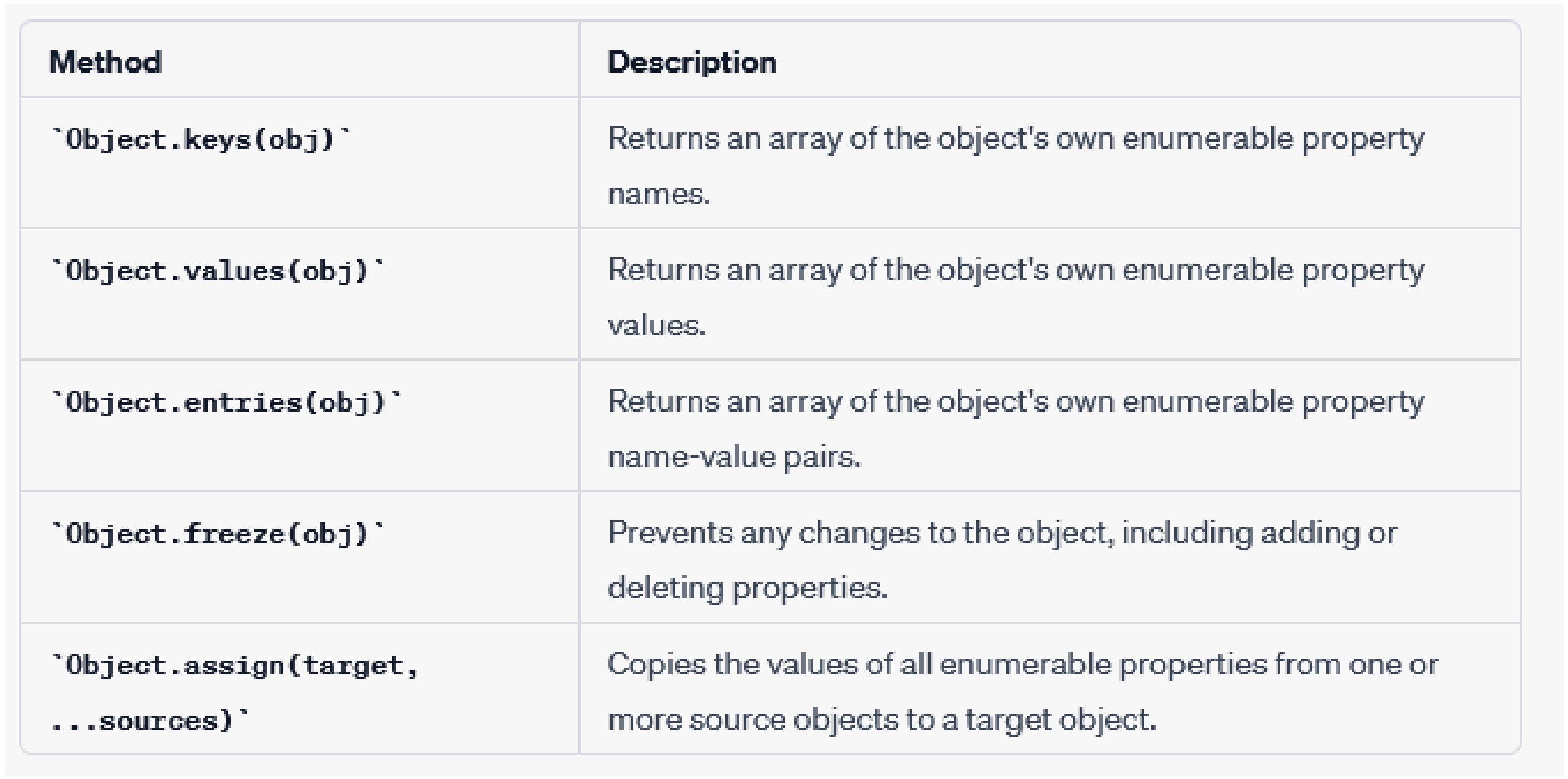
Constructors are functions used to create new instances of an object with the new keyword.



```
// Define a constructor function
function Person(name, age) {
  this.name = name;
  this.age = age;
  //   this keyword in a constructor function
  //   This allows you to set properties and r
}

// Create a new object using the constructor f
const person1 = new Person("Alice", 30);
```

Object Methods

Object has built-in methods that can be used with objects to perform various operations.

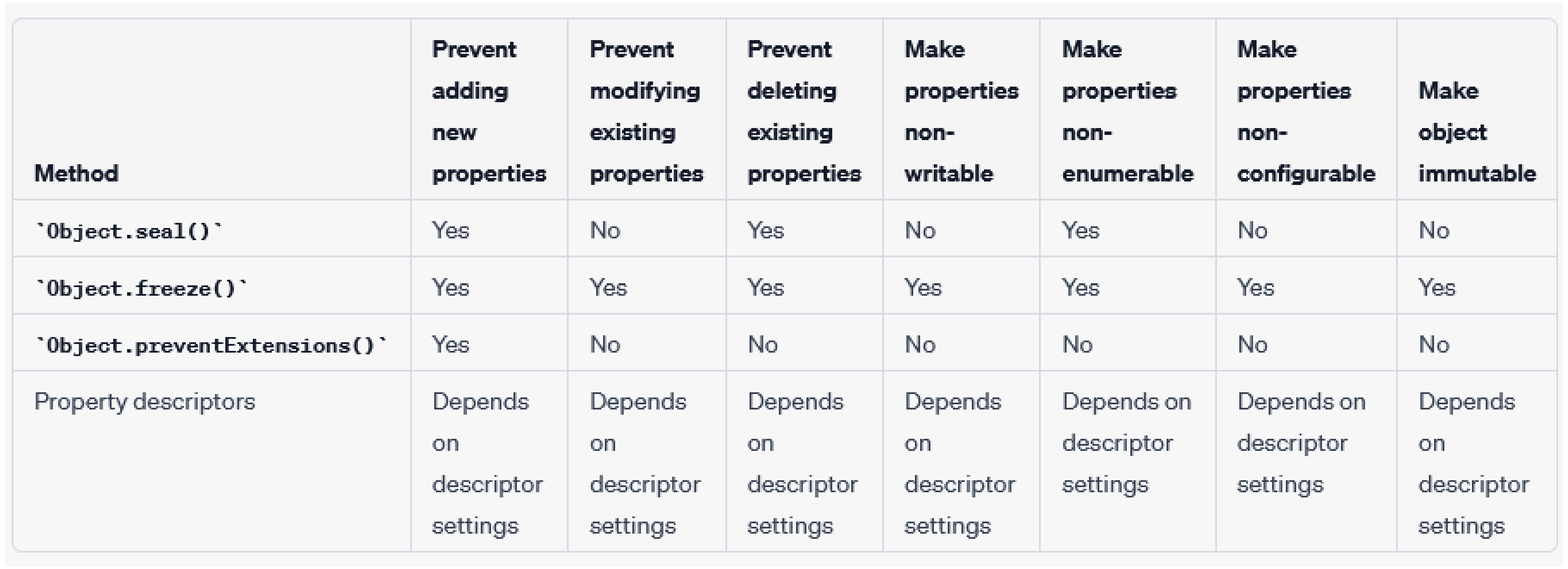


Method	Description
`Object.keys(obj)`	Returns an array of the object's own enumerable property names.
`Object.values(obj)`	Returns an array of the object's own enumerable property values.
`Object.entries(obj)`	Returns an array of the object's own enumerable property name-value pairs.
`Object.freeze(obj)`	Prevents any changes to the object, including adding or deleting properties.
`Object.assign(target, ...sources)`	Copies the values of all enumerable properties from one or more source objects to a target object.

Object Protection

Several ways to protect an object from being modified.

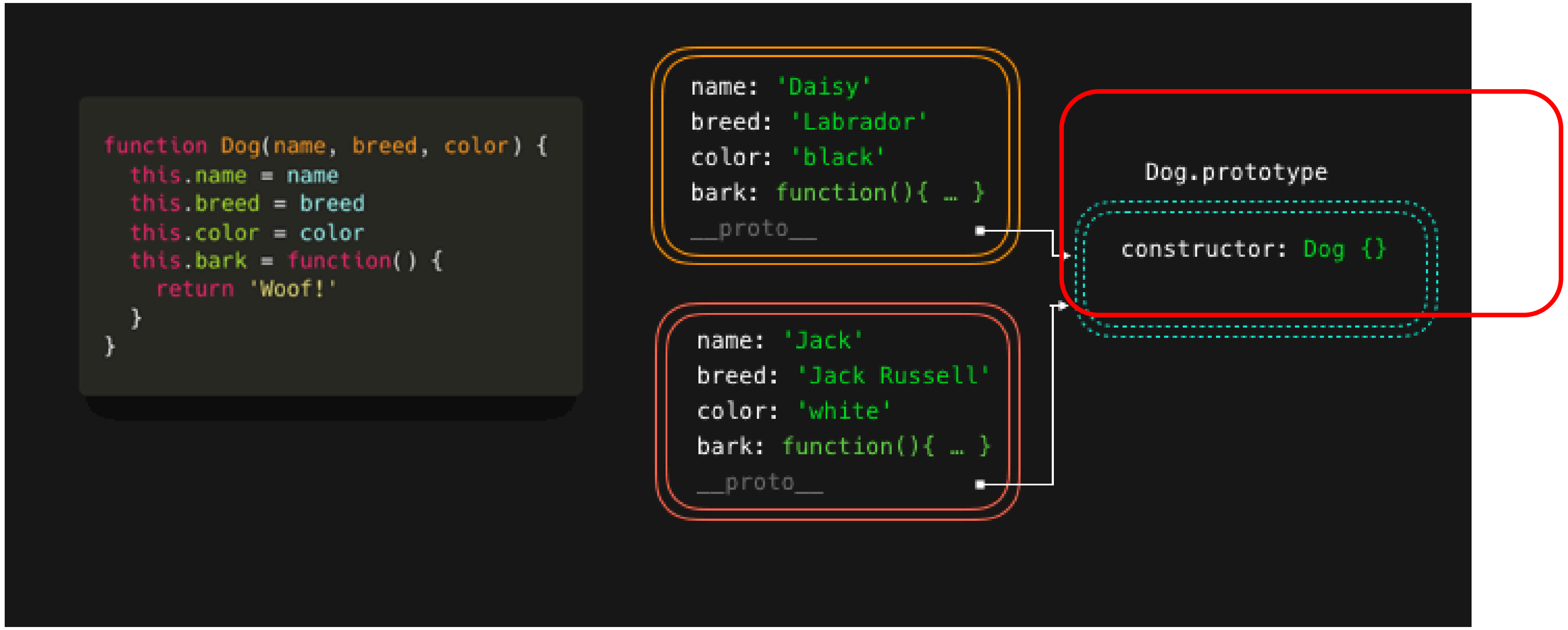
- **Object.seal():** This method seals an object and prevents any new properties
- **Object.defineProperty():** This method allows you to define a property on an object and set various attributes such as writable to false
- **Object.freeze():** This method freezes an object and prevents any modifications



Method	Prevent adding new properties	Prevent modifying existing properties	Prevent deleting existing properties	Make properties non-writable	Make properties non-enumerable	Make properties non-configurable	Make object immutable
`` `Object.seal()` ``	Yes	No	Yes	No	Yes	No	No
`` `Object.freeze()` ``	Yes	Yes	Yes	Yes	Yes	Yes	Yes
`` `Object.preventExtensions()` ``	Yes	No	No	No	No	No	No
Property descriptors	Depends on descriptor settings	Depends on descriptor settings	Depends on descriptor settings	Depends on descriptor settings	Depends on descriptor settings	Depends on descriptor settings	Depends on descriptor settings

Object: Deep Dive

Everything in JavaScript is **Not class based it's prototype based**



```
function Dog(name, breed, color) {
  this.name = name
  this.breed = breed
  this.color = color
  this.bark = function() {
    return 'Woof!'
  }
}
```

```
name: 'Daisy'
breed: 'Labrador'
color: 'black'
bark: function(){ … }
__proto__
```

```
name: 'Jack'
breed: 'Jack Russell'
color: 'white'
bark: function(){ … }
__proto__
```

```
Dog.prototype
constructor: Dog {}
```

All about <Prototype/>

Prototype is an object that **serves as a template or blueprint for other objects**

In JavaScript, **every object has a prototype**

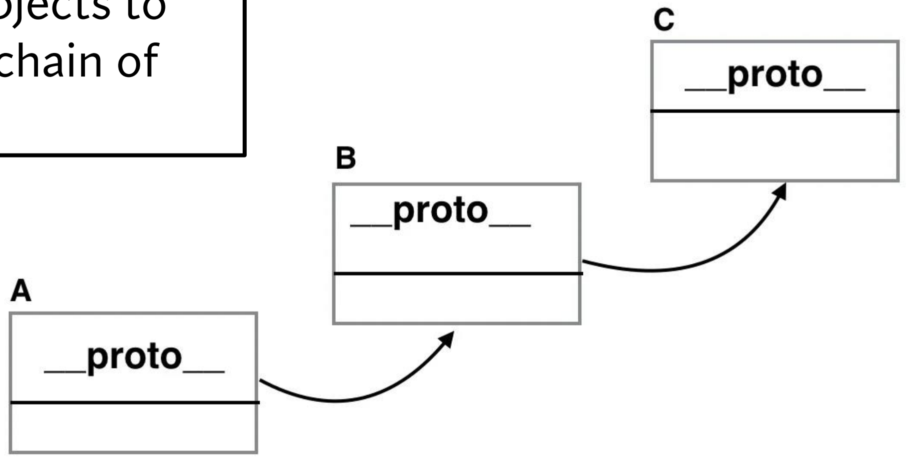
Objects **inherit properties and methods from a prototype.**

Prototype- further..

- You can change/modify prototype of any object
- JavaScript prototype property allows you to add/delete new properties to object constructors

Prototype Chaining

Prototypal chaining is the mechanism in JavaScript that allows objects to inherit properties and methods from their prototypes, creating a chain of inheritance.



Knowledge check

1. How to prevent a Object from being modified further?

Object.freeze/seal/using writable property

2. What is the output of below code?

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}

Person.prototype.getDetails = function() {
  return `Name: ${this.name}, Age: ${this.age}`;
}

const person1 = new Person("John", 30);
const person2 = new Person("Alice", 25);

console.log(person1.getDetails()); // Output:
console.log(person2.getDetails()); // Output:
```

// Output: Name: John, Age: 30
// Output: Name: Alice, Age: 25

What is OOPS?

Object-Oriented Programming (OOP) is a programming paradigm that uses objects to represent and manipulate data.

- **Encapsulation:** The process of hiding the implementation details
- **Inheritance:** The ability of an object to inherit properties and methods from a parent
- **Polymorphism:** The ability of an object to take on many forms
- **Abstraction:** The process of simplifying complex systems by breaking them down into smaller

This OOPS ES5 syntax looks complex and not readable to me!

Hence ES6 introduced Classes

Introduction to ES6

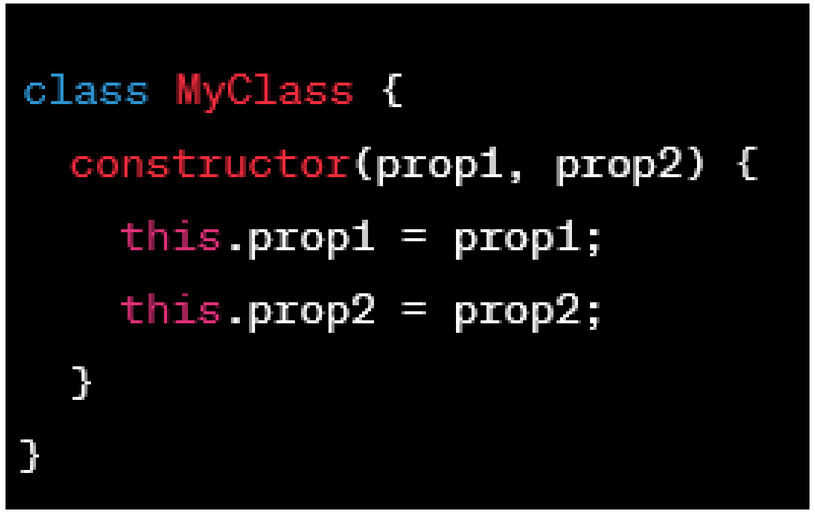
ES6, also known as ECMAScript 2015, brought many updates to the JavaScript language

- Classes
- Arrow functions
- Let and const
- Promises

and many more..

ES6 Classes

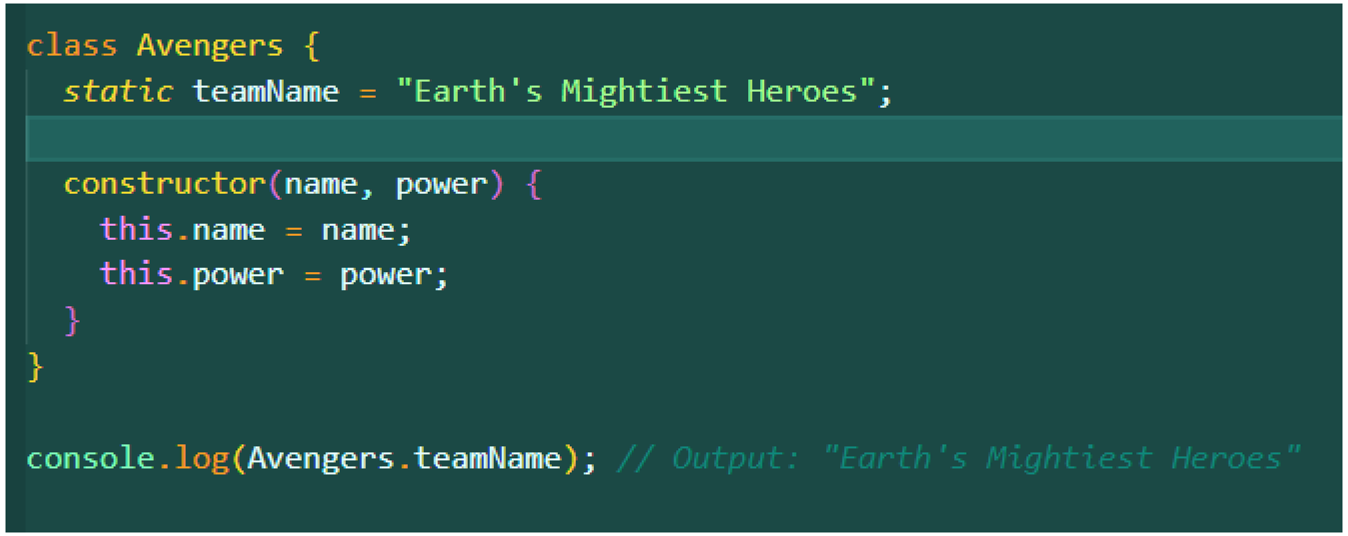
ES6, Class- is a blue print of object (Class is not object)



```
class MyClass {
  constructor(prop1, prop2) {
    this.prop1 = prop1;
    this.prop2 = prop2;
  }
}
```

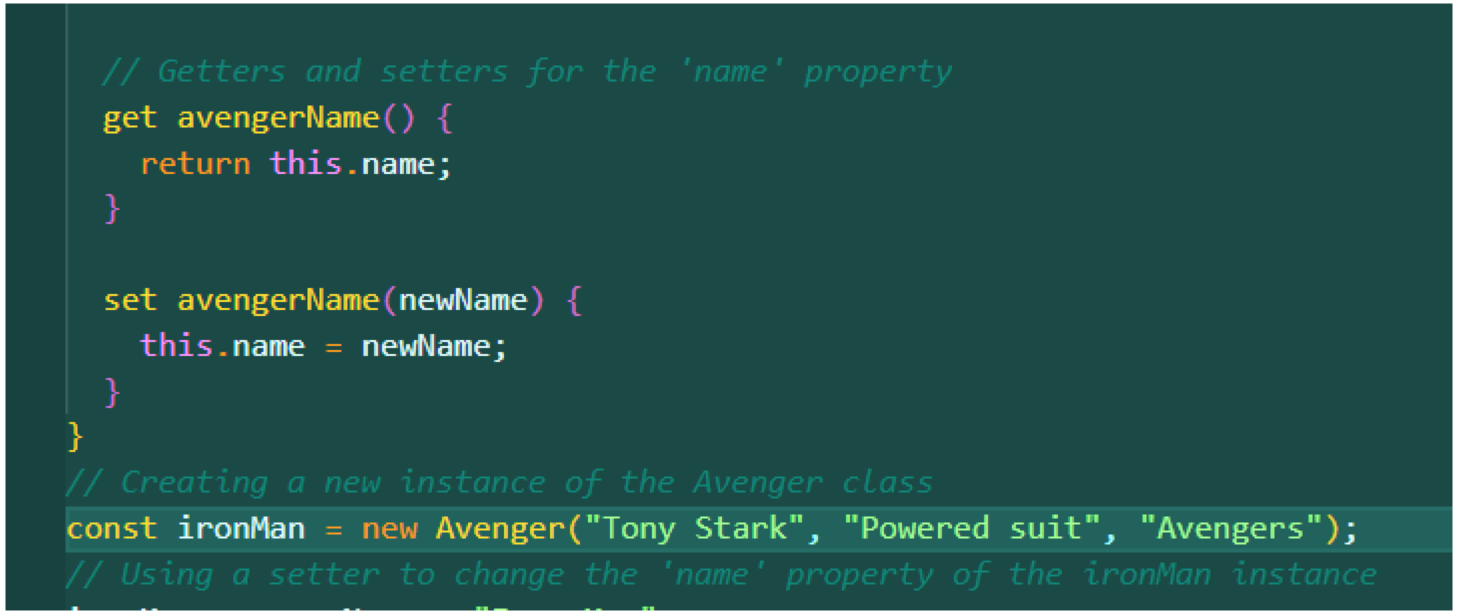
Static Properties

Static Properties are properties **that are defined on the class itself** rather than on its instances. They are **accessed using the class name followed by the property name**



```
class Avengers {
  static teamName = "Earth's Mightiest Heroes";

  constructor(name, power) {
    this.name = name;
    this.power = power;
  }
}

console.log(Avengers.teamName); // Output: "Earth's Mightiest Heroes"
```

Get and Set



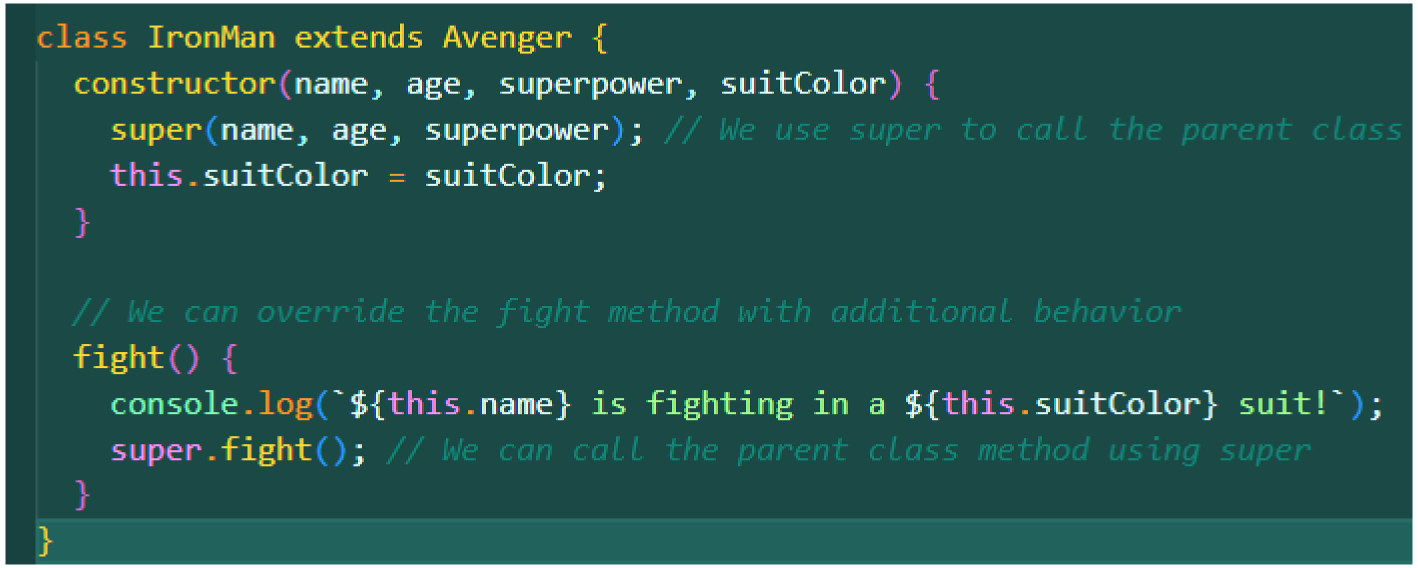
```
  // Getters and setters for the 'name' property
  get avengerName() {
    return this.name;
  }

  set avengerName(newName) {
    this.name = newName;
  }
}
// Creating a new instance of the Avenger class
const ironMan = new Avenger("Tony Stark", "Powered suit", "Avengers");
// Using a setter to change the 'name' property of the ironMan instance
```

OOPS using ES6 Classes

- **Encapsulation:** Process of wrapping similar code in one place
- **Inheritance:** Sub classing
- **Polymorphism:** The ability of an object to take on many forms
- **Abstraction:** is the process of **hiding the internal details** of an application from the outer world.

<'super'/> keyword

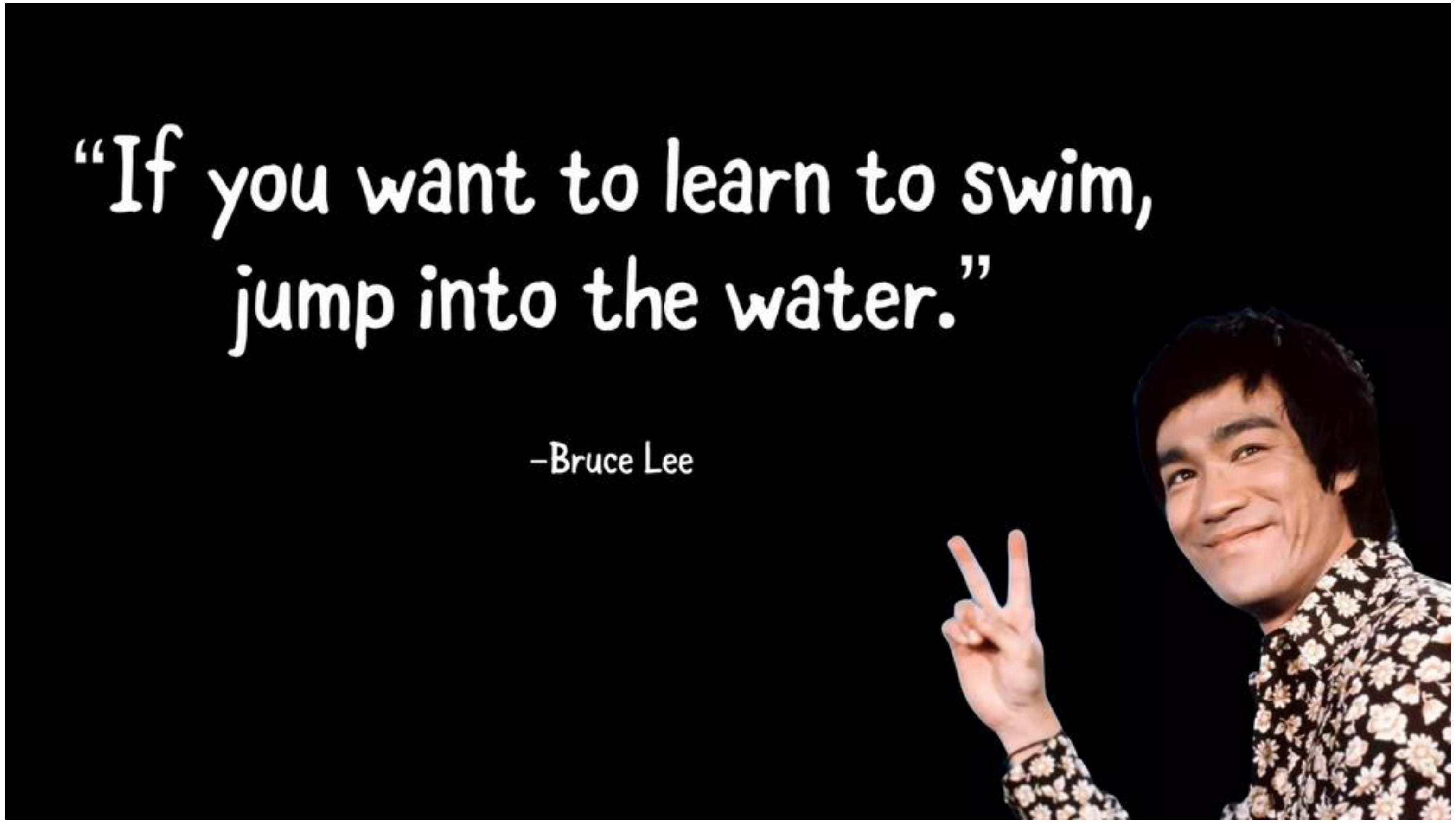
Super() is a keyword used to **call the parent class constructor** or parent class methods **from a subclass.**



```
class IronMan extends Avenger {
  constructor(name, age, superpower, suitColor) {
    super(name, age, superpower); // We use super to call the parent class
    this.suitColor = suitColor;
  }

  // We can override the fight method with additional behavior
  fight() {
    console.log(`${this.name} is fighting in a ${this.suitColor} suit!`);
    super.fight(); // We can call the parent class method using super
  }
}
```

Knowledge check

Difference between Classical Vs Prototypal inheritance


"If you want to learn to swim, jump into the water."
–Bruce Lee

<|>esto

Q & A