

Who am I?

Humera Tariq

PhD, MS, MCS (Computer Science), B.E (Electrical)

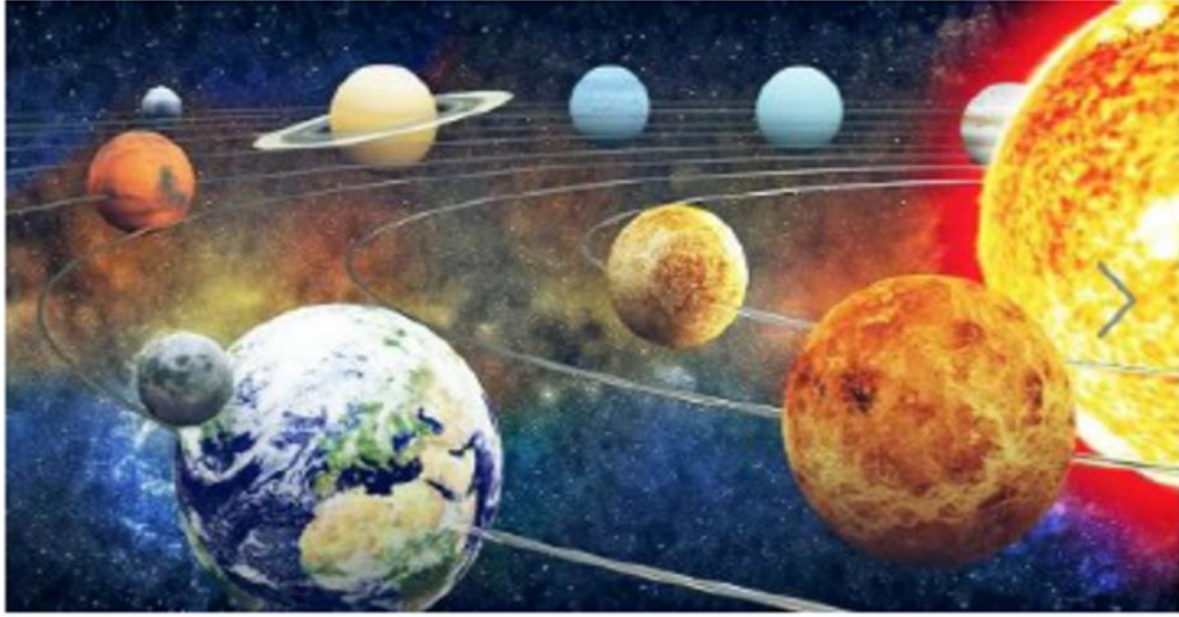
Postdoc (Medical Image Processing, Deep Neural Networks)

Email: humera@uok.edu.pk

Web: <https://humera.pk/>

Discord: <https://discord.gg/xeJ68vh9>

Starting in the name of Allah,



*the most beneficial,
the most merciful.*

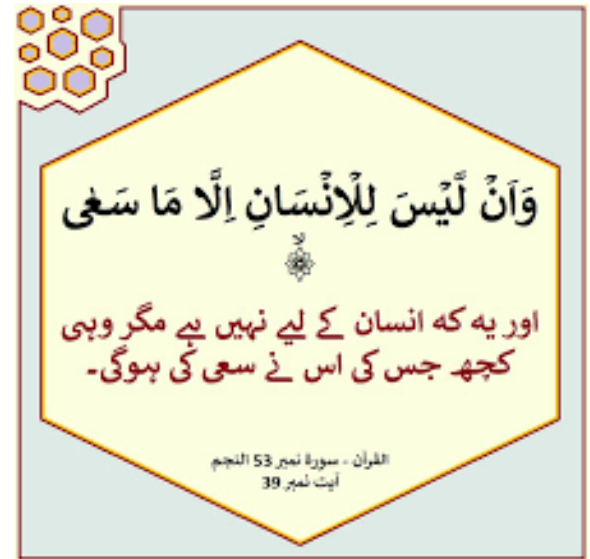
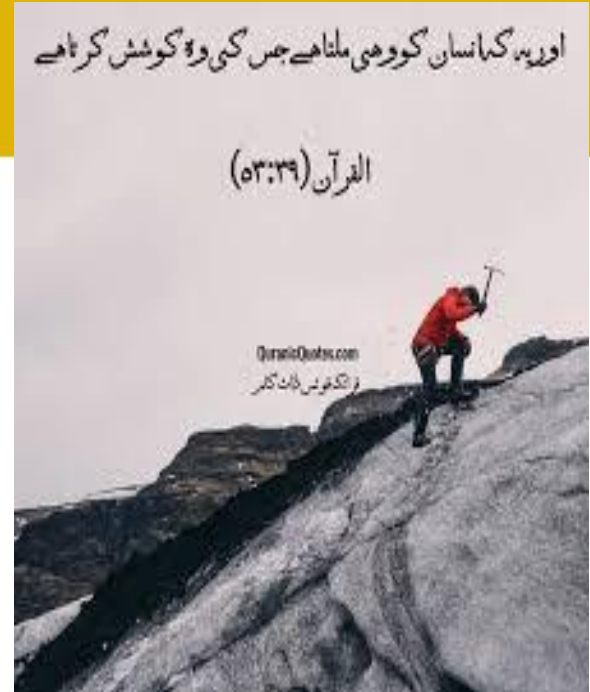
أَمْرٌ لِلْإِنْسَانِ مَا تَبَىٰ
تَبَىٰ ۚ

کیا انسان کو ہر وہ چیز حاصل ہے جس کی اس نے تمنا کی؟



UNIVERSITY OF
KARACHI

And there is not for man except that [good] for which he strives.



Week 05

Internet Application Development



Copyright © 2025, Humera Tariq

Department of Computer Science (DCS/UBIT)
University of Karachi
January 2025

Week 04 Recap.....

- ✓ Java script (JS) run-time
- ✓ Hoisting, Scoping
- ✓ Synchronous vs Asynchronous
- ✓ JS what are you?
- ✓ V8 Engine do you know event loop, call back, http request, apis e.g. `setTimeout`?



What is Node.js?



Node.js is an _____ , _____ **JavaScript runtime environment** that allows developers to execute JavaScript code on the _____ side. It was released in 2009 by Ryan Dahl and is built on the **Chrome V8 JavaScript engine**. Node.js enables the development of **scalable** and **efficient network applications** by allowing JavaScript to run outside of a _____ .

1990

HOW THE WEB BEGAN!

1989

Sir Tim Berners-Lee made his first proposal for the Web to make sharing information across the Internet easier. The proposal wasn't initially accepted!



1991

People outside of CERN (Tim's workplace) primarily University-based scientific departments and physics laboratories were invited to use the Web.



1994

Tim moved to Massachusetts Institute of technology where he founded the World Wide Web Consortium (W3C) devoted to developing and upkeeping open Web standards.

1990



Tim has created the three core technologies of the Web (HTTP, URI and HTML). From which he created the first website.

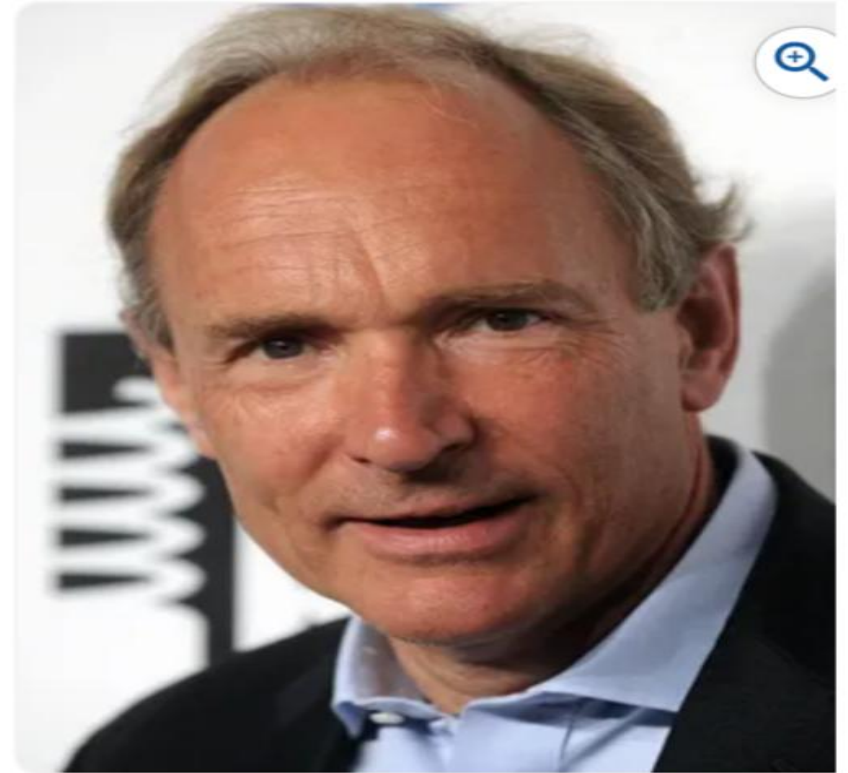


1993

Tim wanted everyone and anyone to have access to the web so it would achieve its full potential. So announced he'd ensure CERN allowed the royalty-free use of the Web forever.



PROMPTPC.CO.UK



Tim Berners-Lee Tim Berners-Lee, 2014.

Quick Facts

In full: Sir Tim Berners-Lee

Born: June 8, 1955, [London, England](#) (age 69)

Founder: [World Wide Web Consortium](#)

Inventions: [World Wide Web](#) • [World Wide Web](#)

2009



This talk has been presented at Node Congress 2023, check out the latest edition of this [JavaScript Conference](#).

Early Life and Inspiration Growing up with a passion for programming, Ryan was intrigued by the idea of JavaScript running outside the browser. This fascination led him to create Node.js, a groundbreaking platform that allows developers to use JavaScript for server-side scripting.

The Birth of Node.js In 2009, Ryan introduced Node.js to the world, unleashing a new era of JavaScript development. His vision was to build a platform that could handle thousands of concurrent connections with minimal overhead, making real-time applications more efficient and scalable.

Impact and Legacy Node.js quickly gained popularity among developers due to its speed and versatility. It became the go-to choice for building fast, scalable network applications, powering everything from web servers to IoT devices.

Continued Innovation Even after stepping away from the project for a few years, Ryan's impact on the Node.js community continues to be felt. His recent work on Deno, a secure runtime for JavaScript and TypeScript, showcases his commitment to pushing the boundaries of web development.

Conclusion Ryan Dahl's journey is a testament to the power of a single idea to transform an entire industry. His innovative spirit and relentless pursuit of excellence continue to inspire developers around the world. As we celebrate the legacy of Node.js, let's also celebrate the visionary behind it all - Ryan Dahl.

NODE.JS



VS

DENO





Pieoneers

2011-2012



React

DID YOU KNOW?

JORDAN WALKE

Genius Behind
Web industry



pieoneerssoftware • [Follow](#)



development.

In 2011, Walke was working at Facebook when he introduced React to tackle the complex challenges of building dynamic, interactive user interfaces. Since then, it has become a cornerstone of modern web development, empowering platforms like Instagram, Airbnb, and countless others. Its component-based architecture and virtual DOM have allowed developers to build faster and more scalable applications.

React has set a new standard in front-end development, making it one of the most popular JavaScript libraries today. Thank you, Jordan, for your invaluable contribution to the tech world!

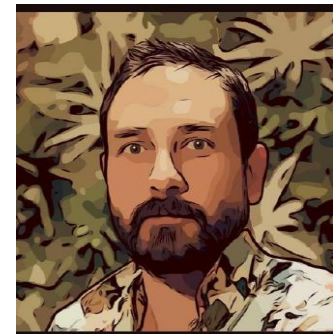
[#ReactJS](#) [#WebDevelopment](#)



5 likes

September 26, 2024

[Log in](#) to like or comment.

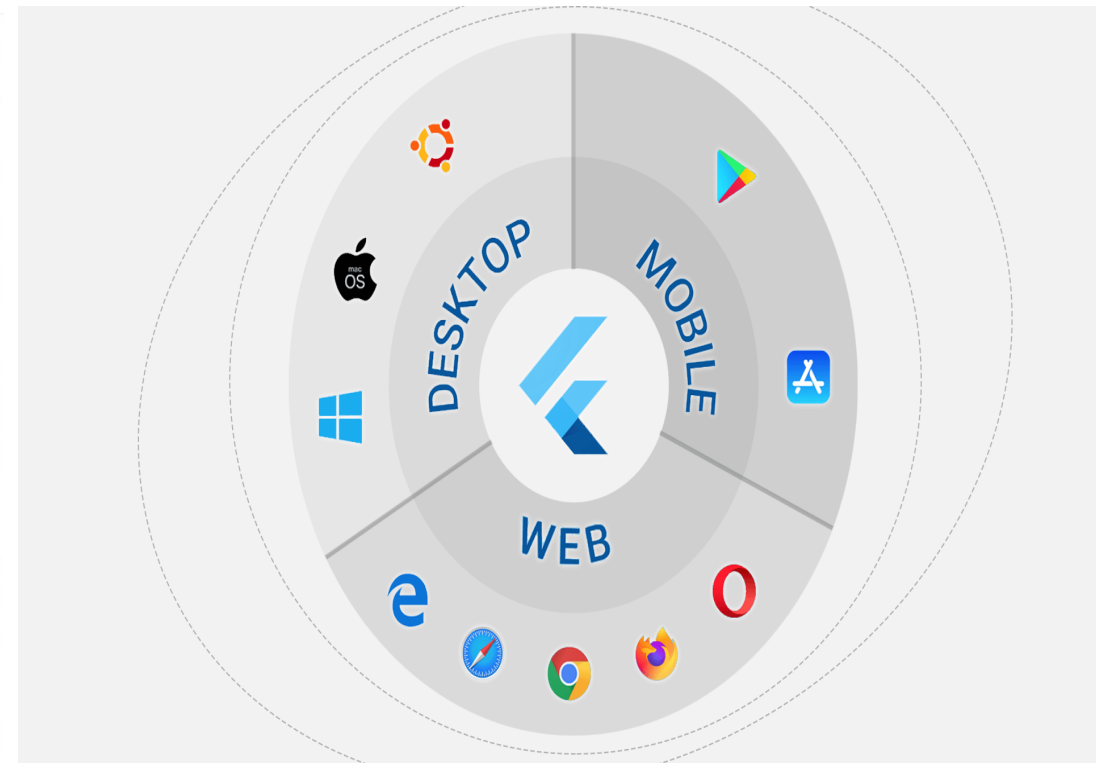
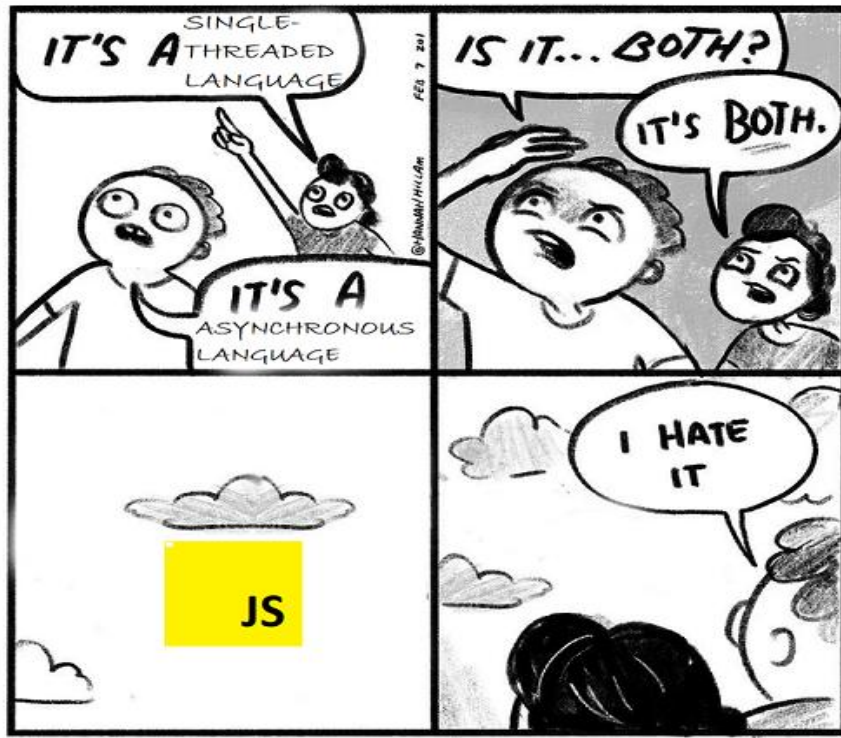
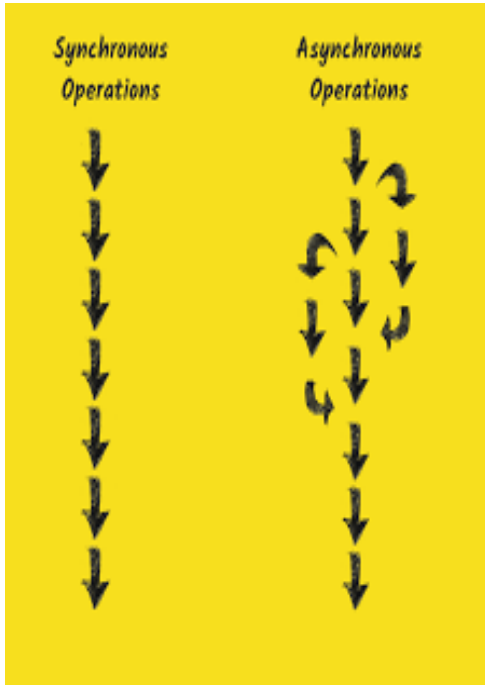


Upcoming event



React Summit 2025

📍 June 13 - 18, 2025. Amsterdam & Online



Key Features of Node.js

- 1.Asynchronous and Event-Driven:** Node.js uses asynchronous programming, which means it can handle multiple requests simultaneously without waiting for any one request to complete. This makes it highly efficient and suitable for real-time applications¹.
- 2.Single-Threaded:** Node.js runs on a single thread using non-blocking I/O calls, which allows it to handle thousands of concurrent connections with minimal overhead.
- 3.Cross-Platform:** Node.js can run on various platforms, including Windows, Linux, Unix, and macOS.
- 4.Rich Ecosystem:** Node.js has a large ecosystem of open-source libraries and modules available through the Node Package Manager (NPM), which simplifies development and enhances functionality

Today's Agenda

- ✓ Three (3) Fundamental Pillars of JS ?
- ✓ Fundamentals of Web-apis ?

Java Script.....

- ✓ Scoping ... loops, call stack, call backs ,event loop
- ✓ Hoisting
- ✓ this
- ✓ Closure

Class Game after ES6!

Game

```
constructor(n) {  
    this.name = n;  
}
```

```
printName() {  
    console.log(this.name);  
}
```

let Keyword

Arrow functions

Multi-Line String

const Keyword

Classes

for...of Loop

Modules

Template Literals

ES6

Rest Parameters

Default Parameters

Destructuring
Assignments

Spread Operator



UNIVERSITY OF
KARACHI

```
JS game.js > ...
1  class Game {
2      constructor(n) {
3          this.name = n;
4      }
5      printName() {
6          console.log(this.name);
7      }
8  }
```



What will be the output?

```
let g1 = new game("Chess");
```

```
let g2 = new game("Football");
```

```
console.log(g1.printName === g2.printName);
```

```
JS game.js > ...  
1   class Game {  
2       constructor(n) {  
3           this.name = n;  
4       }  
5       printName() {  
6           console.log(this.name);  
7       }  
8   }
```

```
let g1 = new Game("Chess");  
let g2 = new Game("Football");  
console.log(g1.printName === g2.printName);
```

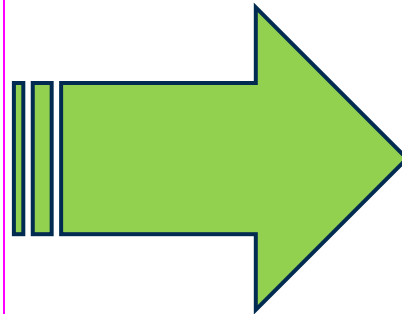
- ✓ The **function inside a class** after ES6 allow **multiple instances to share the same method**, improving memory efficiency.

Does Game(n) Act as a Constructor?



JS game.js > ...

```
1  class Game {  
2    constructor(n) {  
3      this.name = n;  
4    }  
5    printName() {  
6      console.log(this.name);  
7    }  
8  }
```



JS game.js > ...

```
1  class Game {  
2    Game(n) {  
3      this.name = n;  
4    }  
5    printName() {  
6      console.log(this.name);  
7    }  
8  }
```

What will be the output?



```
class Game {  
  Game(n) {  
    console.log("Game method called!");  
    this.name = n;  
  }  
  printName() {  
    console.log(this.name);  
  }  
}
```

```
let g1 = new Game("Chess");  
g1.printName();
```

This proves that `Game(n)` is just a method, your object instances **never call it automatically**. However, because you didn't define a `constructor()`, JavaScript just creates an empty object.

Java Script.....

- ✓ Scoping ... loop, call stack, call backs ,event queues
- ✓ Hoisting
- ✓ this ...constructor ...prototype method, inheritance
- ✓ Closure

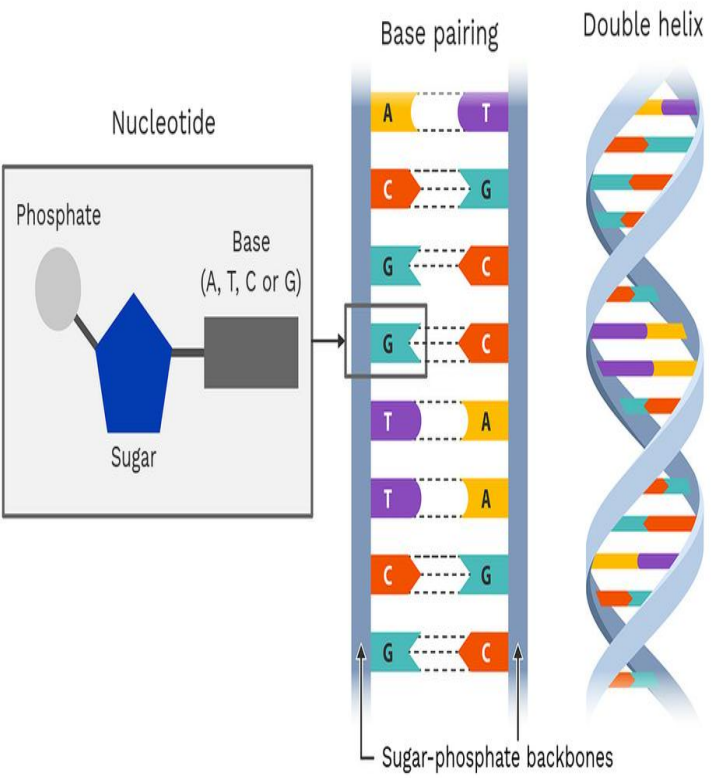
What will be the output?



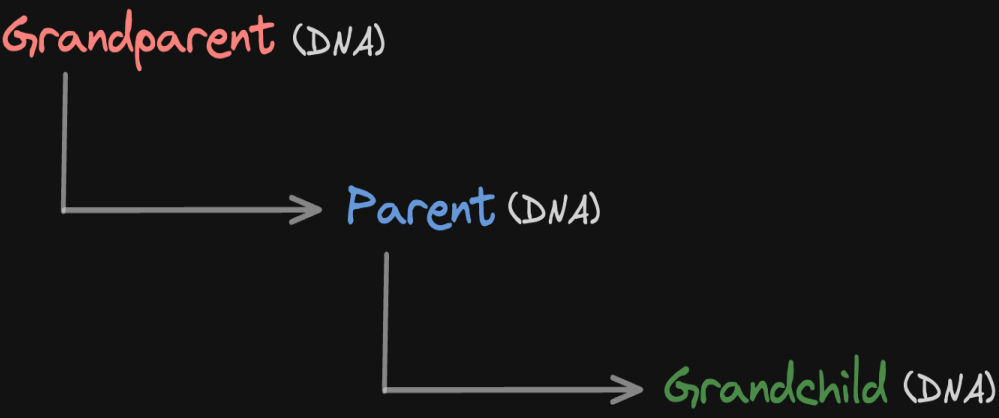
```
class Game {  
  constructor(n) {  
    this.name = n;  
  }  
  
  printName() {  
    console.log(this.name);  
  }  
}  
  
console.log(typeof Game);  
console.log(Game.prototype.printName);
```

(Yes! Class is just a
_____)

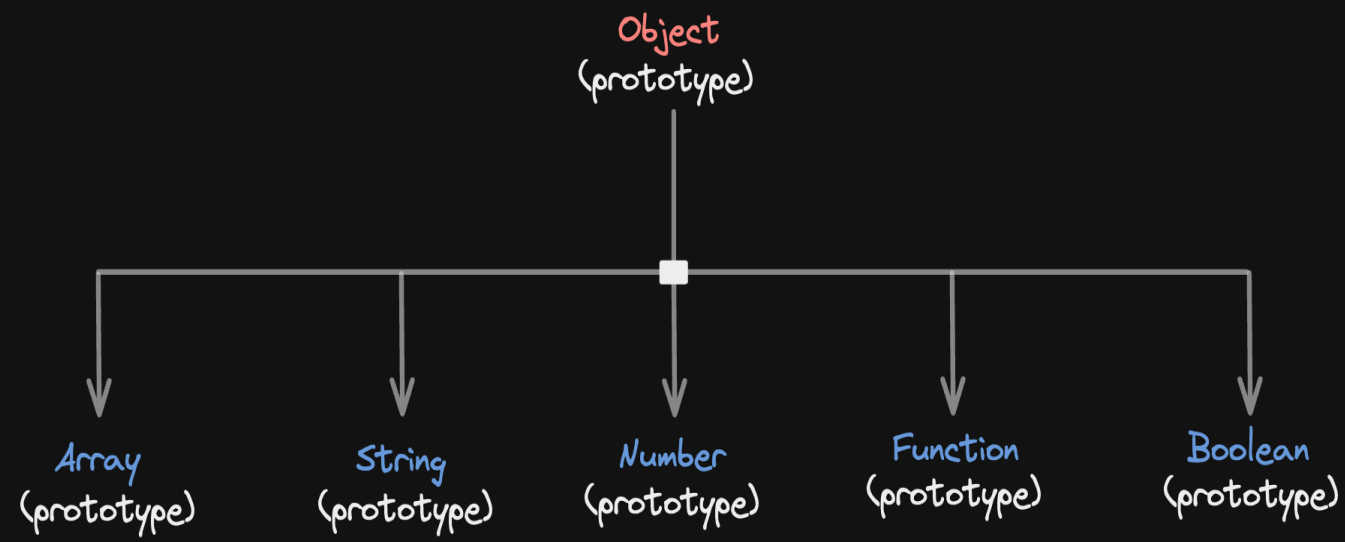
`printName()` is a
_____ **method**, even
though we wrote it inside
a class.



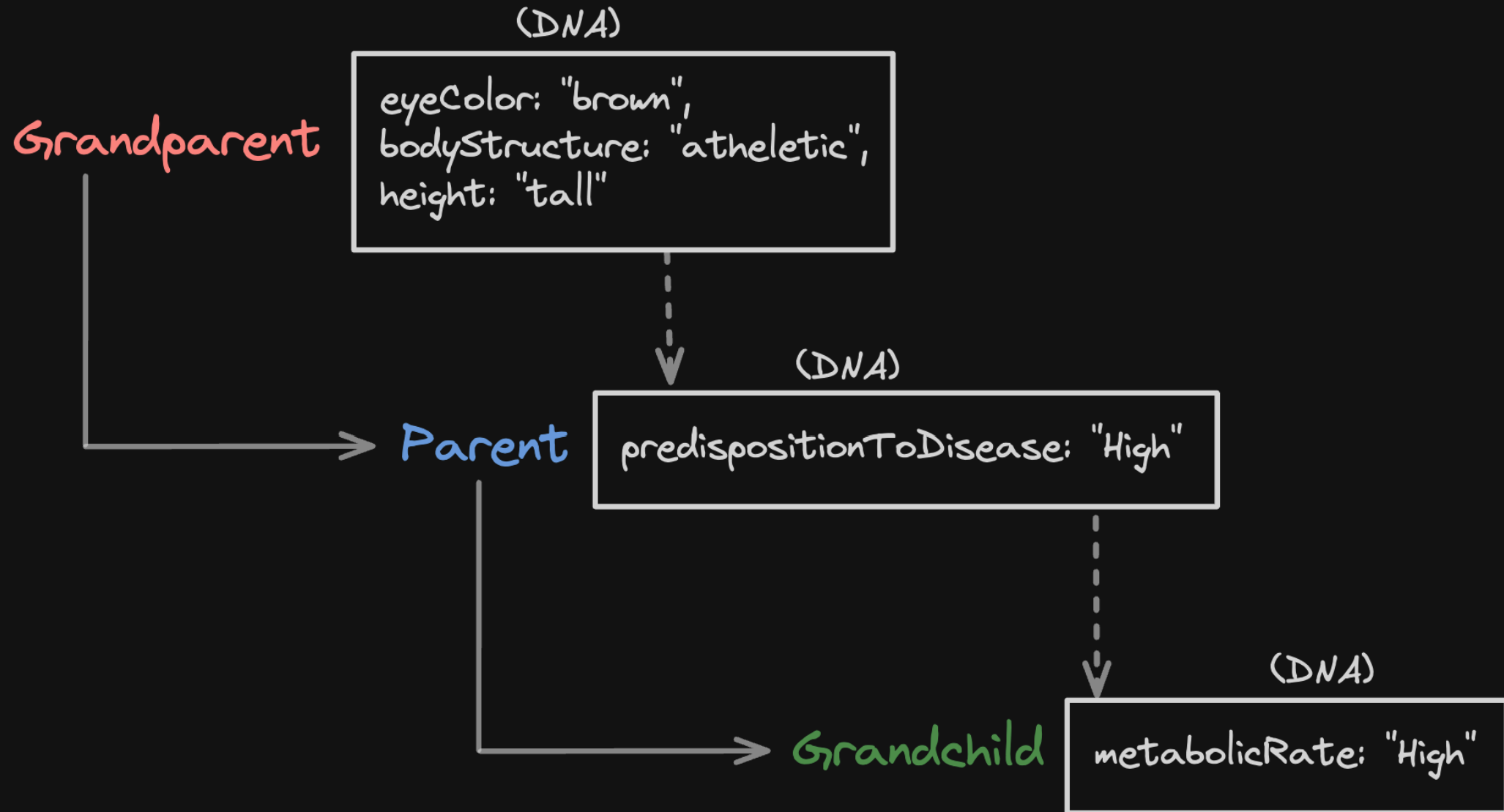
HUMAN INHERITANCE



INHERITANCE IN JAVASCRIPT



HUMAN INHERITANCE

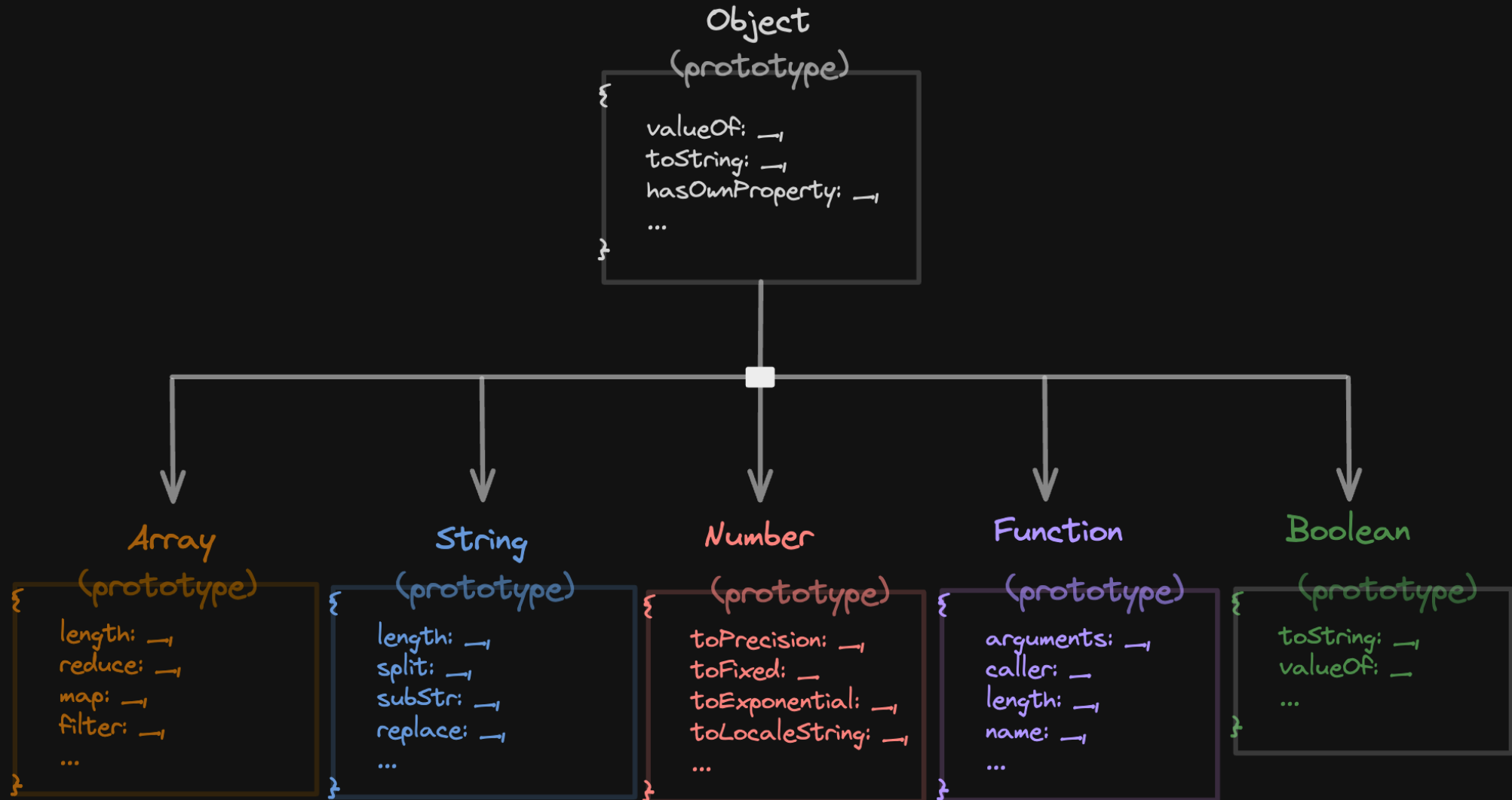


The **properties in the prototype** of a constructor are inherited by the children created by that constructor. This continues down the chain.

You can reason about it like this:

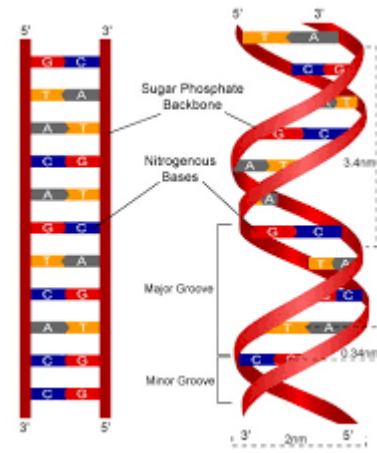
Every descendant in the inheritance chain, inherits everything available in the **prototype of its ancestors.**

PROTOTYPAL INHERITANCE



In JavaScript, **Arrays** and **Functions** are objects and have an internal **[[Prototype]]** property. However, **String**, **Number**, and **Boolean** are **primitive types**, but JavaScript temporarily **wraps them in their object counterparts** (String, Number, Boolean) when accessing properties or methods."

Just like **DNA** acts as **blueprints** that **define** characteristics that are passed on through generations of the human family tree, **prototypes** in JavaScript are used **to define** properties and methods that are inherited by objects down the JavaScript Object tree.



console.log(<Type>.prototype); in JavaScript.

Prototype Logged	Methods Inside)	Who Inherits It?
<code>console.log(Array.prototype);</code>	<code>.push()</code> , <code>.pop()</code> , <code>.map()</code> , <code>.filter()</code> , <code>.reduce()</code>	All arrays (<code>[]</code> , <code>new Array()</code>)
<code>console.log(String.prototype);</code>	<code>.toUpperCase()</code> , <code>.toLowerCase()</code> , <code>.trim()</code> , <code>.charAt()</code> , <code>.includes()</code>	All strings (<code>"hello"</code> , <code>'abc'</code>)
<code>console.log(Number.prototype);</code>	<code>.toFixed()</code> , <code>.toString()</code> , <code>.valueOf()</code> , <code>.toPrecision()</code>	All numbers (<code>42</code> , <code>3.14</code>)
<code>console.log(Function.prototype);</code>	<code>.call()</code> , <code>.apply()</code> , <code>.bind()</code>	All functions (<code>function() {}</code>)
<code>console.log(Boolean.prototype);</code>	<code>.toString()</code> , <code>.valueOf()</code>	All booleans (<code>true</code> , <code>false</code>)



Before **ES6 (2015)**, developers had to **explicitly** use _____ to share methods.



ES6 class introduced a **more readable** way to define _____, making **prototype less visible**.



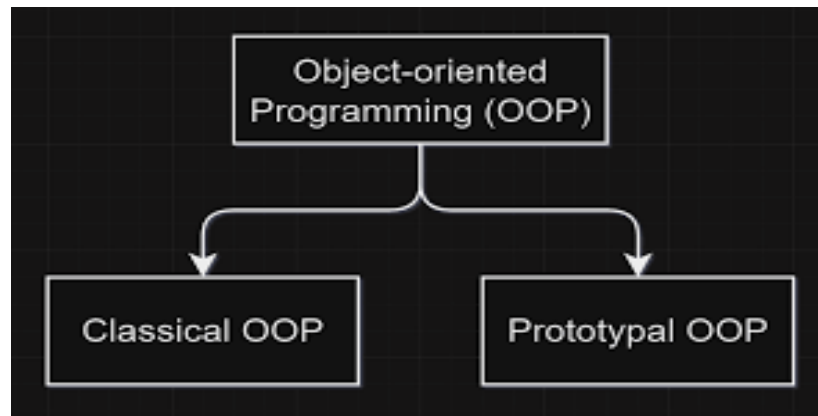
But **internally, JavaScript still uses** _____ and **ES6 class** **update** just hides the _____.



Understanding prototypes **helps you debug and** _____ **JS code better**.



Prototypes allow _____ sharing & _____ efficiency. Without prototypes, every _____ would have its own copy of methods, leading to huge memory waste.



Three (3) main pillars around which the JS language is organized: _____ ,
_____, and _____ .

```
function Game(n) {  
  this.name = n;  
  this.printName = function () {  
    console.log(this.name);  
  };  
}  
  
let g1 = new Game("Chess");  
let g2 = new Game("Football");  
  
console.log(g1.printName === g2.printName);
```



printName is _____ (shared /not shared)



Every time `new Game(...)` is called, a new _____ is created in memory.

```
function Game(n) {  
    this.name = n;  
}  
  
Game.prototype.printName = function () {  
    console.log(this.name);  
};  
  
let g1 = new Game("Chess");  
let g2 = new Game("Football");  
  
console.log(g1.printName === g2.printName);
```

What is a Shared Function (Prototype Method)?

Instead of creating a new function for every _____, we can store it **once** in the _____ and let all instances access the same function.

```
function Game(n) {  
    this.name = n;  
}  
  
Game.prototype.printName = function () {  
    console.log(this.name);  
};  
  
let g1 = new Game("Chess");  
let g2 = new Game("Football");  
  
console.log(g1.printName === g2.printName);
```

How Does JS Find the Function? (Prototype Chain)

When you call `g1.printName()`, JavaScript looks for `printName` in this order:

- ✓ First, it checks `g1` (the instance itself). If `printName` is found inside `g1`, it uses that function.
- ✓ If not found, it looks in `Game.prototype`. If `printName` is found there, it uses it.
- ✓ If not found in `Game.prototype`, it moves up to `Object.prototype`.
- ✓ If still not found, it **returns undefined**.

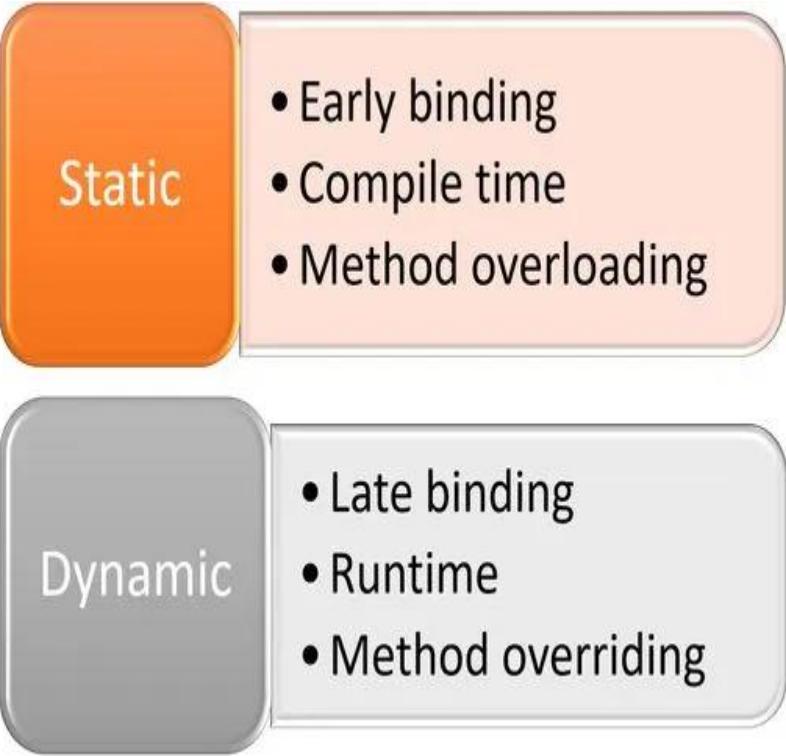
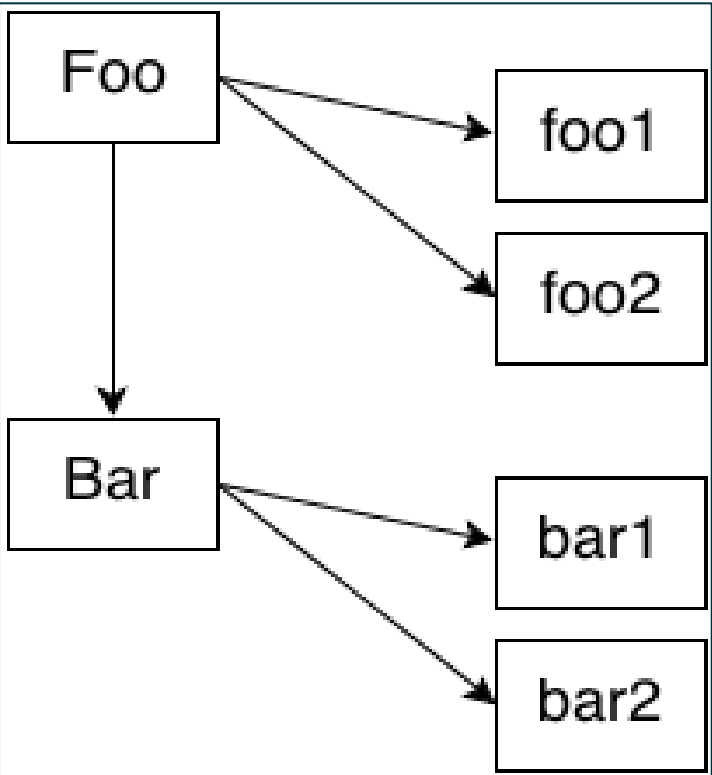
This is called **prototype chaining**.

Approach	Memory Efficient?	Function Sharing	Syntax Readability	Recommended?
(1) ES6 <code>class</code>	✓ Yes	✓ Yes (<code>prototype</code> is used automatically)	✓ Cleanest	✓ Yes
(2) Prototype Method	✓ Yes	✓ Yes (<code>prototype</code> is used explicitly)	✗ Less readable	✓ Yes
(3) Inside Constructor	✗ No	✗ No (each instance has its own copy)	✓ Readable but inefficient	✗ No

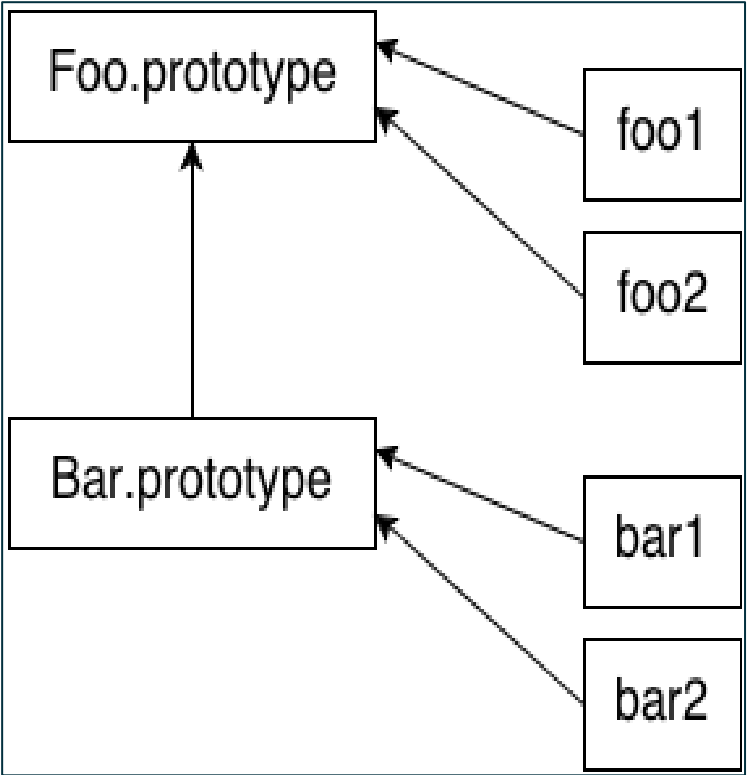
Key Differences Between ES5 (Prototype) and ES6 (Class)

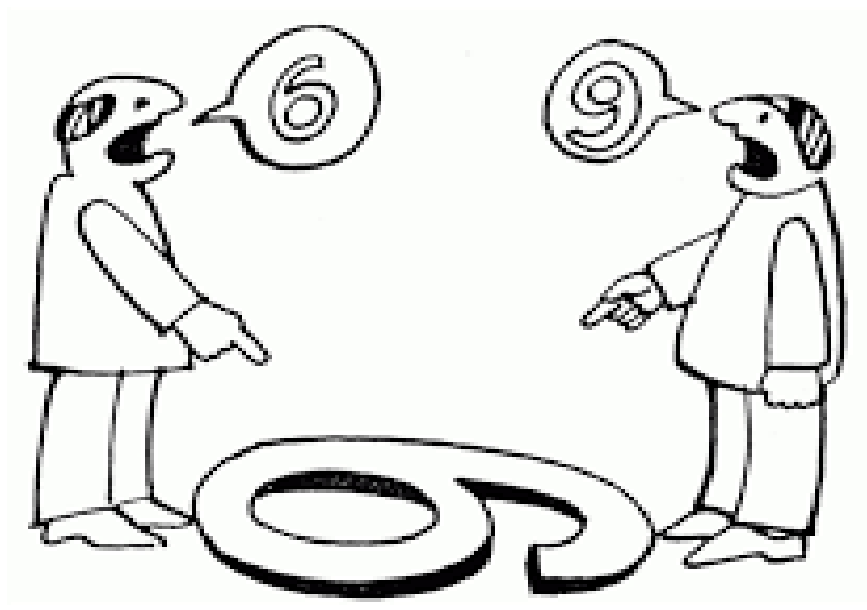
Feature	ES5 (Prototype Method)	ES6 (Class Method)
Syntax	More manual, explicit	More readable, structured
Method Location	<code>Game.prototype.method</code>	Automatically in <code>Game.prototype</code>
Inheritance	Manual (<code>Object.create</code> , <code>call</code> , <code>apply</code>)	<code>extends</code> and <code>super</code> make it simpler
Readability	Harder to read	Cleaner and more intuitive

"prototypal inheritance"



"Behavior delegation"





Some devs insist that "delegation" is just the dynamic version of "inheritance", like two sides of the same coin, but I see them as **orthogonal systems**.




THANKS
for your
ATTENTION

Any
questions



UNIVERSITY OF
KARACHI



"Don't be satisfied with stories, how things have gone with others. Unfold your own myth." ~Rumi



UNIVERSITY OF
KARACHI



Department of Compute Science (UBIT Building), Karachi, Pakistan.

1200 Acres (5.2 Km sq.)

53 Departments

19 Institutes

25000 Students



My Homeland Pakistan

