Constructor function properties

> in any constructor function if we add any variable inside that using this will it got attached to prototype of that function or not

Excellent question, Vipul 👋 — this touches a very **important concept** in JavaScript: the difference between **instance properties** and **prototype properties**.

Let's break it down clearly and step-by-step.

💡 Short Answer:

> ❌ No — if you add a variable or method using **`this` inside a constructor function**, it does **NOT** get attached to the prototype.
> ✅ It gets attached to the **individual object (instance)** created from that constructor.

🔍 Example:

```
js


function Counter() {
  this.count = 0;  // added via `this`
  this.increment = function() {
    this.count++;
  };
}

const c1 = new Counter();
const c2 = new Counter();
```

Now let's inspect:

```
js


console.log(c1.hasOwnProperty("count"));       // ✅ true
console.log(c1.hasOwnProperty("increment"));   // ✅ true
console.log(Counter.prototype.hasOwnProperty("increment")); // ❌ false
```

✅ Explanation:

- Every time you use **`new Counter()`**,
  → A new object is created.
  → All properties declared with **`this`** (like **`this.count`**, **`this.increment`**) are **copied into that object**.
- So **`c1`** and **`c2`** each get **their own copies** of **`count`** and **`increment`**.
  They do **not share** them via the prototype.

🧠 If you attach to the prototype:

If you want to share the same method among all instances (more memory-efficient), you attach it like this:

```
js

function Counter() {
  this.count = 0;  // instance property
}

Counter.prototype.increment = function() {
  this.count++;
  console.log(this.count);
};
```

Now:

```
js

const c1 = new Counter();
const c2 = new Counter();

console.log(c1.hasOwnProperty("increment"));          // ❌ false
console.log(Counter.prototype.hasOwnProperty("increment")); // ✅ true
```

✅ **`increment`** is shared between all objects via the prototype
✅ **`count`** is unique to each instance

🔄 Summary Table

Where property is defined	Example	Belongs to	Shared among all instances?
Inside constructor with **`this`**	**`this.count = 0`**	Instance	❌ No
On prototype manually	**`Counter.prototype.increment = ...`**	Prototype	✅ Yes

Would you like me to also show how this works in **classes** (which are just syntactic sugar over prototypes)?