Lecture-13

Class Syntax

DOM

Coding Blocks - Kartik Mathur

Class Agenda

Class Syntax

1

Classes are a template for creating objects. They encapsulate data with code to work on that data. Classes in JS are built on prototypes but also have some syntax and semantics that are unique to classes.

Introduced in 2015, before this JS didn’t had class keyword.

Classes

How to make our own constructor function?

```
let makePerson = {
   print(name, age) {
       console.log(`Hello, I am ${this.name} with ${this.age}`)
   }
}

function Person(name, age) {
   let p = Object.create(makePerson);
   p.age = age;
   p.name = name;
   return p;
}
```

```
let p = Person("Kartik",25);
let p1 = Person("Monu",22);
console.log(p)
console.log(p1)
```

Classes

In JS CLASSES are just Syntactic Sugar and not actual classes!

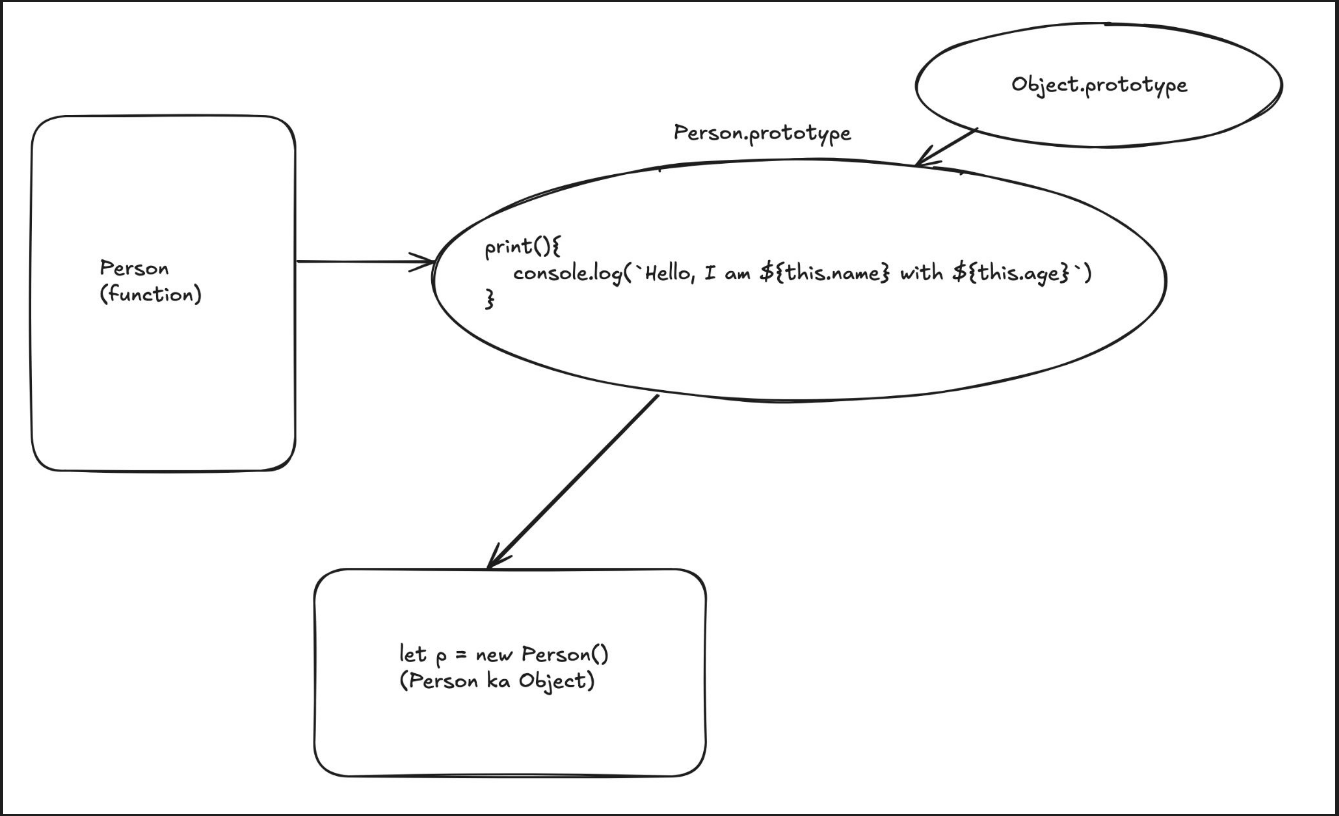
```
class Person{
   constructor(name,age){
       this.name = name;
       this.age = age;
   }

   print(){
       console.log(`Hello, I am ${this.name} with ${this.age}`)
   }
}
```

```
let p = new Person("Kartik",25);
let p1 = new Person("Monu",22);
console.log(p)
console.log(p1)
```

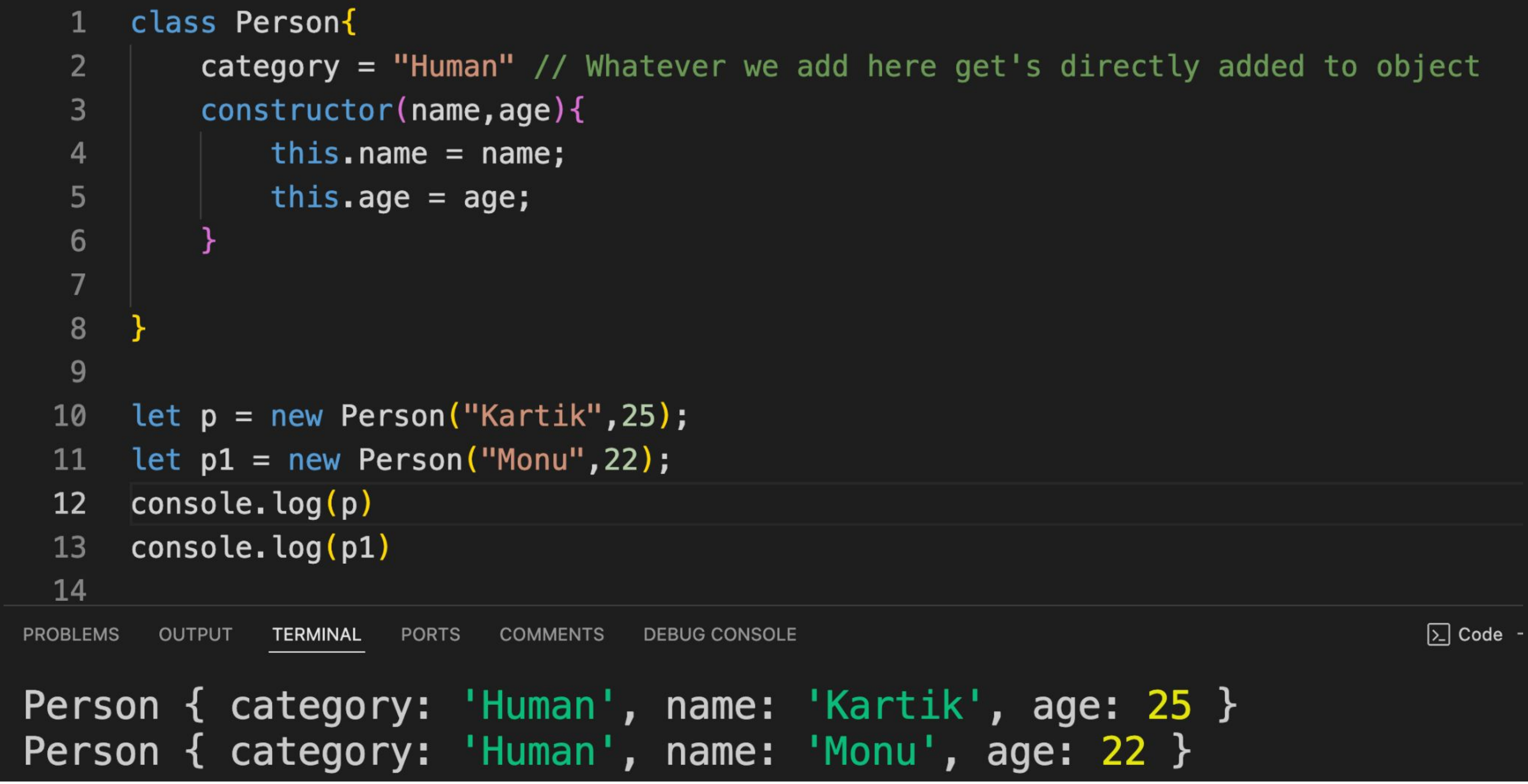
Classes

What it does?



```
print(){
   console.log(`Hello, I am ${this.name} with ${this.age}`)
}
```

Common Properties for all!



```javascript
class Person{
    category = "Human" // Whatever we add here get's directly added to object
    constructor(name,age){
        this.name = name;
        this.age = age;
    }

}

let p = new Person("Kartik",25);
let p1 = new Person("Monu",22);
console.log(p)
console.log(p1)

```

```
Person { category: 'Human', name: 'Kartik', age: 25 }
Person { category: 'Human', name: 'Monu', age: 22 }
```

Private data members in class!

Private data isn’t stored in prototype.
It is stored by JS and managed internally.

```
class Person{
   category = "Human"
   #secret = "My secret";
   constructor(name,age){
       this.name = name;
       this.age = age;
   }

   getSecret(){
       return this.#secret;
   }
}
```

```
let p = new Person("Kartik",25);
console.log(p.getSecret())
console.log(p.#secret) // Error
```

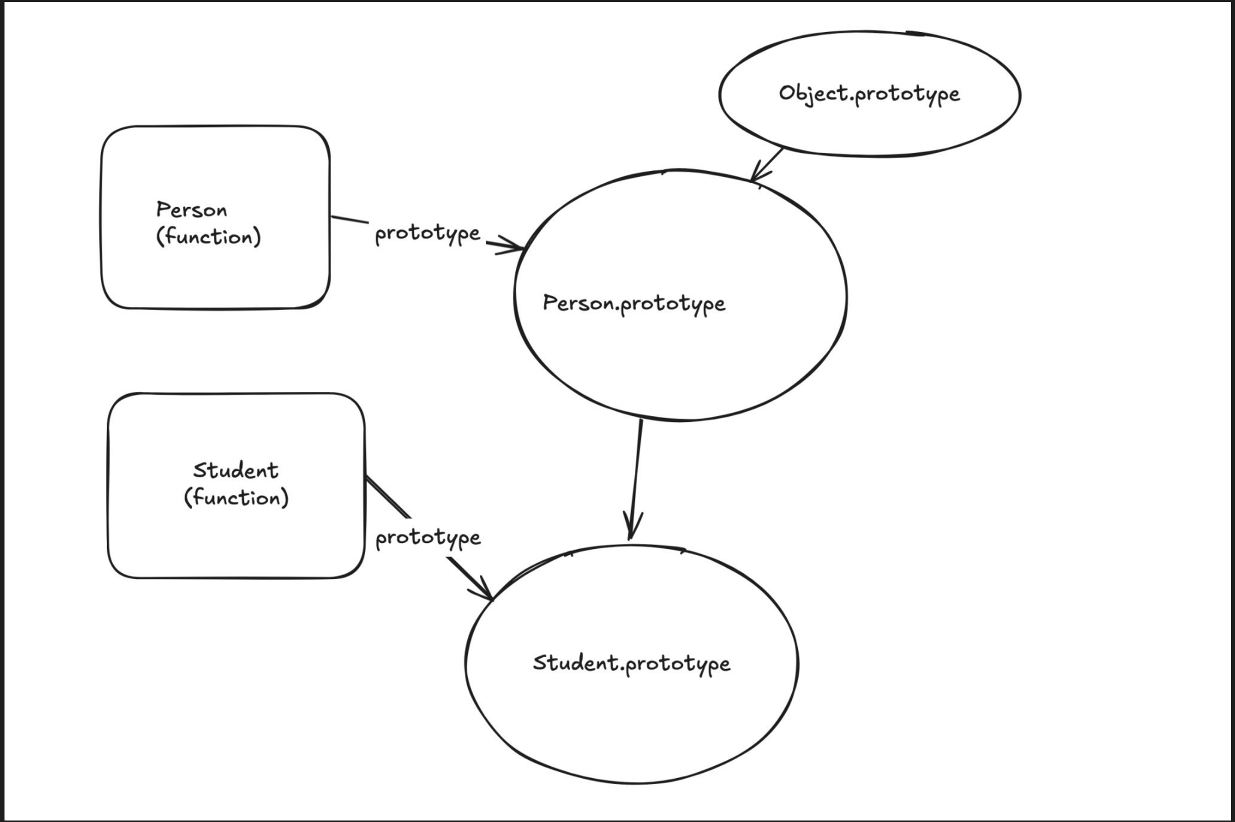
Inheritance and its meaning in classes.

```
class Person{
   constructor(name,age){
       this.name = name;
       this.age = age;
   }
}
class Student extends Person{
   constructor(name,age,marks){
       super(name,age);
       this.marks = marks
   }
}
```

```
let s = new Student("Vaibhav",20,78);
console.log(s);
```

Internal architecture?

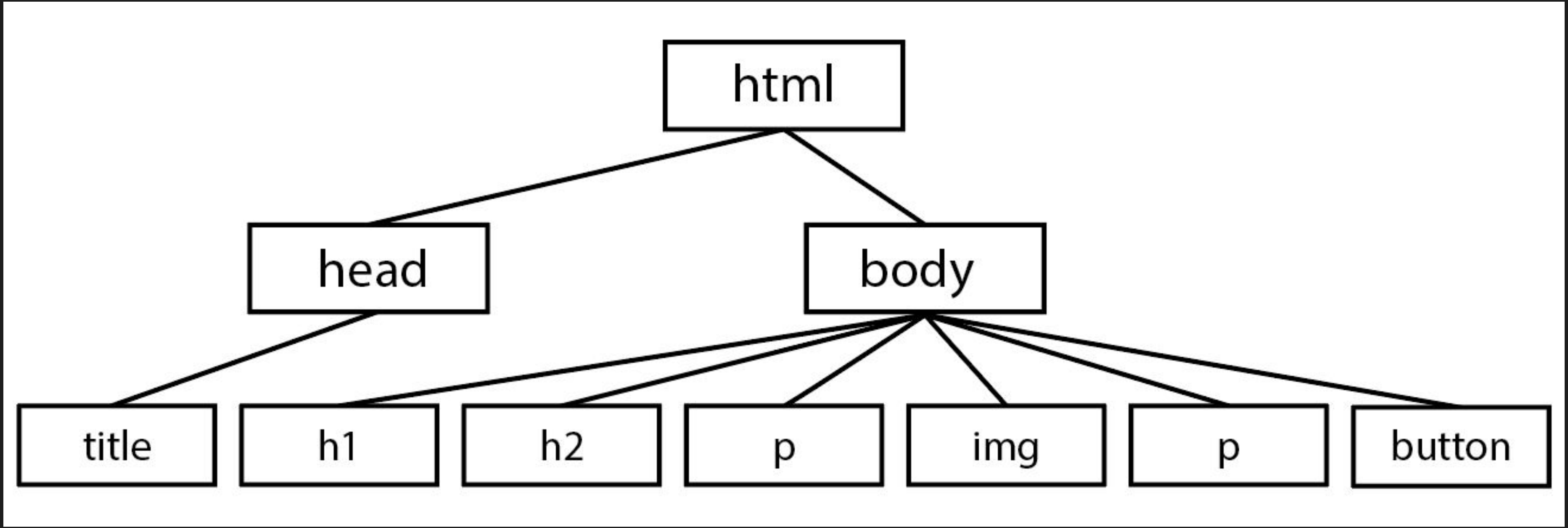
It simply means linking of prototypes! Nothing else (^_^)



DOM Manipulation

2

What is DOM?
A simple HTML tree structure is called as DOM. We can access it with the help of ‘document’ object.



What is the DOM?

- The Document Object Model (DOM) represents the structure of a webpage.
- It allows JavaScript to interact with HTML and CSS dynamically.
- Modify elements, change styles, handle events, and create dynamic content.

1. Access an Element.

- `document.getElementById()`
- `document.getElementByClassName()`
- `document.querySelector()`
- `document.querySelectorAll()`

To Change content or access content:

- innerText
- innerHTML

To Change the CSS:

- `document.querySelector("h1").style.color = "blue";`

Adding or Removing Class:

- `document.querySelector("h1").classList.add("heading");`
- `document.querySelector("h1").classList.remove("heading");`

Create an Element.

- document.createElement()

Append an Item

- document.appendChild()

Accessing Children/Parent/Sibling

EVENT LISTENERS!

JavaScript allows handling events like clicks, mouse movements, key presses, etc.
Two common ways to attach events:

1. `onClick` (Inline or DOM Property Event)
2. `addEventListener` (Event Listener Method)

What is `onClick`?

- Directly assigns an event handler to an element.
- Can only have **one** function assigned at a time (overwrites previous ones).
- Simple but **not flexible** for multiple events.

```
const btn = document.querySelector("button");
btn.onclick = function () {
   console.log("Button clicked!");
};
```

What is `addEventListener`?

- Allows adding multiple event listeners to the same element.
- More **flexible** and supports different event types.
- Can be **removed** using `removeEventListener`.

```
const btn = document.querySelector("button");
btn.addEventListener("click", () => {
   console.log("Button clicked!");
});
```

Overwriting Issue in `onClick`

- `onClick` **overwrites** the previous event.

```
btn.onclick = () => {
   console.log("First event");
};
btn.onclick = () => {
   console.log("Second event"); // This will replace the first one!
};
```

`addEventListener` **allows multiple handlers**.

```
btn.addEventListener("click", () => {
    console.log("First event");
});
btn.addEventListener("click", () => {
    console.log("Second event"); // Both will execute!
});
```

Event Listeners: To do a task on some event.

```
btn.addEventListener('click', () => {
   console.log("Button clicked!");
});
```

Some common events:

dblclick – Fires when an element is double-clicked.
mouseenter – Fires when the mouse enters an element.
mouseleave – Fires when the mouse leaves an element.
mouseover – Fires when the mouse enters an element or its child elements.
mouseout – Fires when the mouse leaves an element or its child elements.
keydown – Fires when a key is pressed.
keyup – Fires when a key is released.
input – Fires when the value of an input field changes.
focus – Fires when an input field is focused.
blur – Fires when an input field loses focus.
removeEventListener – Removes an attached event listener.
change – Fires when the value of an input/select changes.