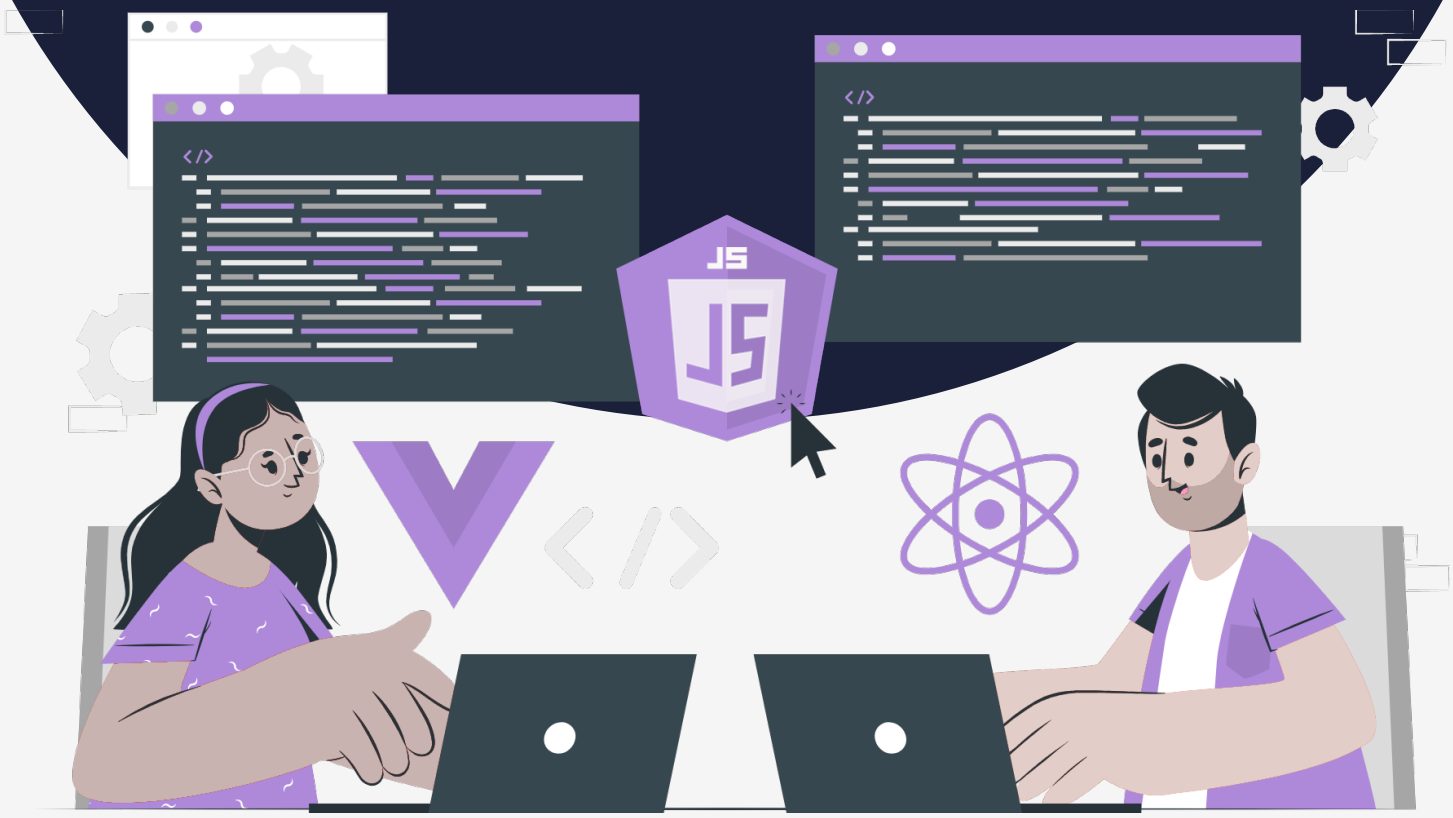
Lesson:

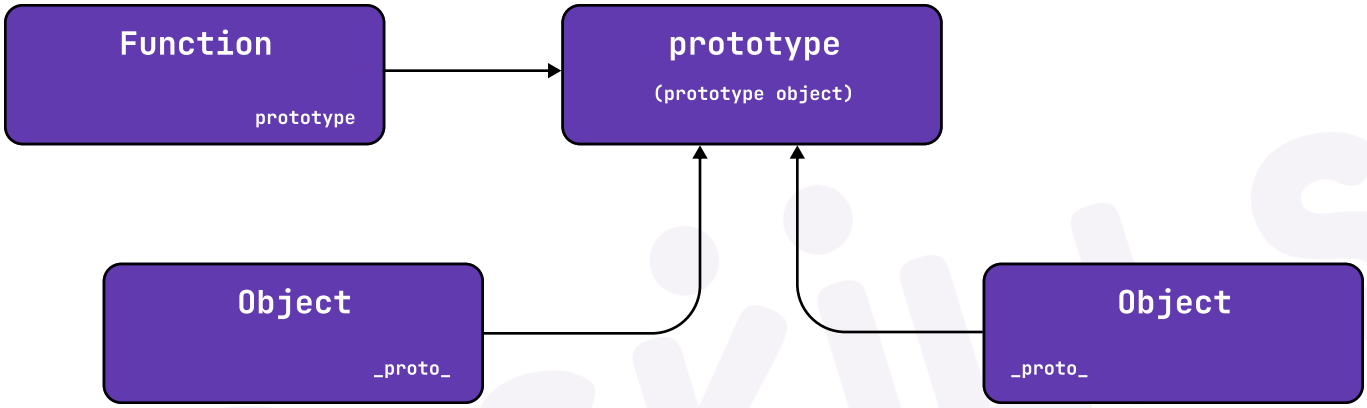
How prototypes Work



PW SKILLS

List of Content:

In JavaScript, everything is an object, including functions. When you create a function in JavaScript, it automatically gets a prototype object. This prototype object is just an ordinary JavaScript object that contains properties and methods that will be shared by all instances of the function.



Prototype object

When you create an instance of a function using the **`new`** keyword, JavaScript sets the new object's internal **`[[Prototype]]`** property to be a reference to the prototype object of the function. This prototype object is often referred to as the constructor's prototype.

proto

When you create a new object of the function using new keyword JS Engine creates an object and sets a property named __proto__ which points to its function's prototype object

Example:
For example, consider the following constructor function:

```
function Person(name, age) {

  this.name = name;

  this.age = age;

}
```

When you create a new **`Person`** object using new **`Person( 'Alice' , 30 )`**, JavaScript sets the **[[Prototype]]** property of the new object to be a reference to the **`Person.prototype`** object:

```
const alice = new Person( 'Alice', 30 );
console.log(alice.__proto__ === Person.prototype); // true
```

This means that the **`alice`** object has access to all of the properties and methods defined on the **`Person.prototype`** object.

Let's add a method to the Person.prototype object:

```
Person.prototype.sayHello = function() {

  console.log('Hello, my name is ' + this.name);

};
```

Now, any **`Person`** object you create will have access to the **`sayHello`** method through its **`[[Prototype]]`** chain:

```
const bob = new Person('Bob', 35);
bob.sayHello(); // Hello, my name is Bob
```

When you call **`bob.sayHello()`**, JavaScript first checks whether the bob object has a **"sayHello()"**. Since it doesn't, JavaScript then checks the **`[[Prototype]]`** property of **`bob`**, which points to the Person.prototype object. JavaScript finds the **`sayHello`** method on the **`Person.prototype`** object and calls it with **`this`** set to the **`bob`** object.

If you add a **`sayHello`** method directly to the **`bob`** object, it will override the method on the prototype:

```
bob.sayHello = function() {

  console.log('Hello from Bob!');

};
bob.sayHello(); // Hello from Bob!
```

Now, when you call **`bob.sayHello()`**, JavaScript finds the sayHello method directly on the **`bob`** object and calls it. The **`sayHello`** method on the **`Person.prototype`** object is ignored.

Prototype Inheritance:

Prototype inheritance refers to the ability of an object to inherit properties and methods from its prototype. When an object is created in JavaScript, it automatically inherits properties and methods from its prototype. If a property or method is not defined on the object itself, JavaScript will look for it on the object's prototype, and if it finds it there, it will use it.

Consider a Person constructor function that creates an object with a name property and a **`sayHello`** method

```
let person = new Person('John');
person.sayHello(); // output: Hello, my name is John
```

In this example, the **sayHello** method is defined on the Person prototype. When we create a new object using the Person constructor function, the object inherits the **sayHello** method from its prototype.

Prototype chaining:

Prototype chaining refers to the way that JavaScript looks for properties and methods when they are not defined on an object. When a property or method is called on an object, JavaScript first checks to see if the property or method is defined on the object itself. If it is not defined on the object, JavaScript then looks for it on the object's prototype. If the property or method is still not found, JavaScript continues to look up the prototype chain until it either finds the property or method or reaches the end of the chain.

In JavaScript, the prototype chain is created automatically when an object is created. The prototype of an object is set to the constructor function's prototype property. This means that any properties or methods defined on the constructor function's prototype will be inherited by all instances of the object created with that constructor function.

Now, suppose we want to create a Student constructor function that creates an object with a name property, a **`studentId`** property, and a **`sayHello`** method. We can do this by using the Person constructor function as a prototype for the Student constructor function:

```
function Student(name, studentId) {
  this.name = name;
  this.studentId = studentId;
}

Student.prototype = new Person();

Student.prototype.sayHello = function() {
  console.log(`Hello, my name is ${this.name} and my student ID is ${this.studentId}`);
};
```

In this example, we set the Student prototype to be a new Person object using the Person constructor function. This means that the Student prototype inherits all the properties and methods from the Person prototype, including the **sayHello** method.

We also redefine the sayHello method on the Student prototype to include the **studentId** property.
Now, we can create a new object student using the Student constructor function and call the **sayHello** method on it:

```
let student = new Student('Jane', '12345');
student.sayHello(); // output: Hello, my name is Jane and my student ID
is 12345
```

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

Person.prototype.sayHello = function() {
  console.log(`Hello, my name is ${this.name}`);
};
```

Now, we can create a new object person using the Person constructor function and call the **sayHello** method on it:

```
let person = new Person('John');
person.sayHello(); // output: Hello, my name is John
```

In this example, when we call the `sayHello` method on the `student` object, JavaScript first looks for the `sayHello` method on the `student` object itself. Since the `sayHello` method is not defined on the `student` object, JavaScript then looks for it on the `Student` prototype. Since the `sayHello` method is defined on the `Student` prototype, JavaScript uses it.

If the `sayHello` method was not defined on the Student prototype, JavaScript would continue to look up the prototype chain until it found the method, or until it reached the end of the chain.

Conclusion

prototypes allow you to define properties and methods that are shared by all instances of a constructor function. When you call a method on an object, JavaScript first looks for the method on the object itself. If it doesn't find the method, it looks for it on the object's prototype, and so on up the prototype chain until it reaches the Object.prototype object, which is the final link in the chain.