Class Methods

This lesson teaches us how to define methods inside a class and explains the use of get/set methods in classes.

WE'LL COVER THE FOLLOWING

- Defining Methods in a Class
  - Example
    - Explanation
- Get/Set Methods
  - Example
  - Explanation

Defining Methods in a Class

In the previous lesson, the `employee` class `constructor` encapsulated all properties and methods. However, methods can also be defined outside the `constructor` in a class.

Whenever a method is declared inside a class, it gets defined on the **prototype** of that class. Hence, whenever an object instance accesses a method, it gets taken from the respective class's *prototype*.

Let's take a look at how that is done.

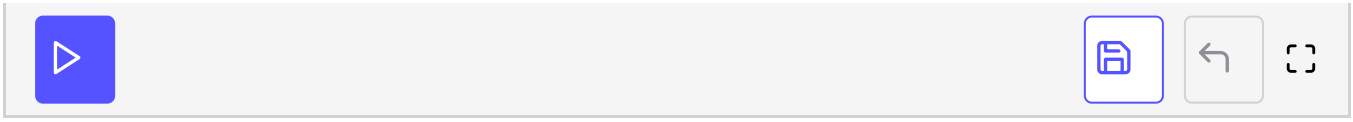
Example

The following example demonstrates how to define methods outside of the `constructor` in a class:

```
//creating a class named employee
class employee{
  //creating the constructor function
  constructor(name,age,designation){
    //all properties defined as they were in the constructor function
    this.name = name
```

```
    this.age = age
    this.designation = designation
    this.displayName = function() {

      console.log("Name is:",this.name)
    }
  }
  //defining methods in a class
  //getAge method returning the age of the current object
  getAge(){
    return this.age
  }
}
//creating an object instance named "employeeObj"
var employeeObj = new employee('Joe',22,'Developer')
//displaying the properties of employeeObj
employeeObj.displayName()
console.log("Age is:",employeeObj.getAge()) //calling the getAge function
console.log("Designation is:",employeeObj.designation)
```



Explanation

The `getAge` function is being defined outside of `constructor` function in **line 15**. All such methods are stored in the *prototype* object of `employee`. A new object, such as `employeeObj`, has access to all the methods defined in the class. When called by `employeeObj`, the method `getAge` is taken from `employee.prototype`.

Get/Set Methods

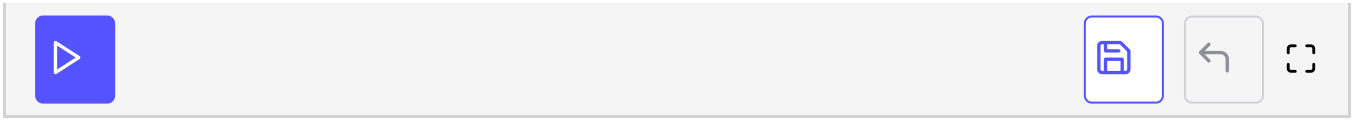
Get/Set *keywords* were discussed previously; they can also be used in classes to get property values.

Example

Let's take a look at an example below:

```
//creating a class named employee
class employee{
  //creating the constructor function
  constructor(name,age,designation){
    //all properties defined as they were in the constructor function
    this.name = name
    this.age = age
    this.designation = designation
  }
  //defining methods in a class
  //getname method returning the name of the current object
  get getname(){
```

```
    return this.name
  }
  //setname method setting the name
  //and displaying the name and the number of alphabets in the name
  set setname(val){
    this.name = val
    console.log("New name is:", this.name)
    console.log(`The name ${this.name} has ${val.length} alphabets`)
  }
}
//creating an object instance named "employeeObj"
var employeeObj = new employee('Joe',22,'Developer')
//displaying the properties of employeeObj
console.log("Name is:",employeeObj.getname)
console.log("Designation is:",employeeObj.designation)
console.log("Age is:",employeeObj.age)
employeeObj.setname = "Ted"
```



Explanation

Similar to the other methods defined in a class, both `getname` and `setname` will be defined in the `employee.prototype` object.

- The `getname` method returns the `name` property of the current object.
- The `setname` method updates the value of `name` in the current object and displays both the new `name` and the number of alphabets in it.

In the next lesson, let's discuss how to protect the properties defined in a class.