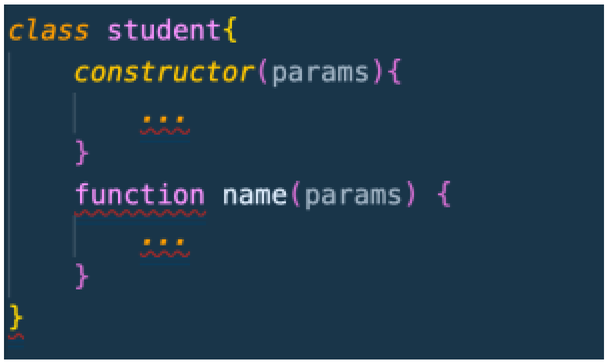
Javascript OOPs

Class

Classes are a unique kind of function in JavaScript. The class can be defined in the same way as function declarations and expressions are. The body of the JavaScript class contains a variety of class members, such as constructors or methods. Strict mode is used when executing the class. Therefore, an error is thrown by the code that contains the silent error or mistake. The class syntax contains two components:

1. Class Declarations - A class can be defined by using a class declaration. A class keyword is used to declare a class with any particular name. According to JavaScript naming conventions, the name of the class always starts with an uppercase letter.
2. Class Expressions - Using a class expression is an additional method of class definition. The class name need not be specified in this case. The class expression is thus capable of being named or unnamed. We can retrieve the class name using the class expression. However, class declaration will prevent this from happening.

Syntax:



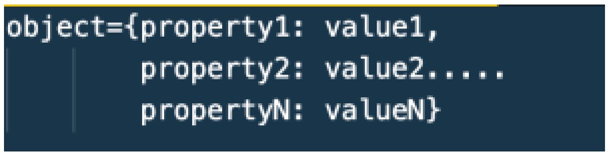
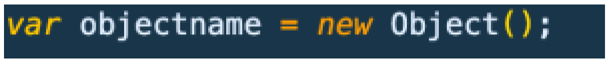
```
class student{
    constructor(params){
        ...
    }
    function name(params) {
        ...
    }
}
```

Objects

A javaScript object is a state-and-behavior-containing entity (properties and method). Examples include a car, pen, bicycle, chair, glass, keyboard, and monitor. JavaScript relies on templates rather than classes. To obtain the object in this case, no class is created. But, we direct create objects.

Creating Objects in Javascript

There are three ways to create objects -

S. No	Types	Description or Syntax
1	By object literal	 `object={property1: value1, property2: value2..... propertyN: valueN}`
2	By creating instance of Object directly (using new keyword)	 `var objectname = new Object();`
3	By using an object constructor (using new keyword)	Here, we have create function with arguments. Each argument value can be assigned in the current object by using this keyword.

Javascript Object Methods

Below are the various Javascripts Object Methods -

S. No	Methods	Description
1	Object.assign()	This method is used to copy enumerable and own properties from a source object to a target object
2	Object.create()	This method is used to create a new object with the specified prototype object and properties.
3	Object.defineProperty()	This method is used to describe some behavioral attributes of the property.

4	Object.defineProperties()	This method is used to create or configure multiple object properties.
5	Object.entries()	This method returns an array with arrays of the key, value pairs.
6	Object.freeze()	This method prevents existing properties from being removed.
7	Object. getOwnPropertyDescriptor()	This method returns a property descriptor for the specified property of the specified object.
8	Object. getOwnPropertyDescriptors()	This method returns all own property descriptors of a given object.
9	Object.getOwnPropertyNames()	This method returns an array of all properties (enumerable or not) found.
10	Object.getOwnPropertySymbols()	This method returns an array of all own symbol key properties.
11	Object.getPrototypeOf()	This method returns the prototype of the specified object.
12	Object.is()	This method determines whether two values are the same value.
13	Object.isExtensible()	This method determines if an object is extensible
14	Object.isFrozen()	This method determines if an object was frozen.
15	Object.isSealed()	This method determines if an object is sealed.
16	Object.keys()	This method returns an array of a given object's own property names.
17	Object.preventExtensions()	This method is used to prevent any extensions of an object.
18	Object.seal()	This method prevents new properties from being added and marks all existing properties as non-configurable.
19	Object.setPrototypeOf()	This method sets the prototype of a specified object to another object.
20	Object.values()	This method returns an array of values.

Prototype

A prototype-based language like JavaScript makes it easier for objects to inherit features and characteristics from one another. Each object in this scene is a prototype. Every time a function is created in JavaScript, the prototype property is immediately added. This property contains a function Object() { [native code] } property and is a prototype object. Each object in JavaScript has a prototype object that inherits its properties and functions. Once again, a prototype object for an object may include a prototype object with additional properties and methods, and so on. It is comparable to **prototype chaining**.

Requirement of Prototype - When a JavaScript object is formed, the associated functions are loaded into memory. Therefore, each time an object is generated, a new copy of the function is created. All the items in a prototype-based approach serve the same purpose. This disregards the need to duplicate the function for each item. The functions are thus initially placed into memory.

Constructor

A unique kind of method called a JavaScript function Object() { [native code] } method is used to initialise and create an object. When memory is assigned to an object, it is called.

Keypoints:

1. The constructor keyword is used to declare a constructor method.
2. The class can contain one constructor method only.
3. JavaScript allows us to use parent class constructor through super keyword.

Static

Instead of a class instance, JavaScript offers static methods that are part of the class. Therefore, calling the static method does not require an instance. Direct calls to these methods are made to the class itself.

Keypoints:

1. A static method is declared with the static keyword.
2. Any name can be used for the static method.
3. More than one static method can be found in a class.
4. If we declare multiple static methods with the same name, JavaScript will always call the most recent one.
5. Utility functions can be written using the static technique.
6. This keyword can be used to invoke a static method inside of another static method.
7. The non-static method cannot directly call a static method using this keyword. In this situation, we may either use the class name or a function Object() { [native code] } property to call the static function.

Pillars of OOPs

Encapsulation

The JavaScript Encapsulation technique involves tying together the functions that operate on the data (variables). It enables us to validate and govern the data. To achieve an encapsulation in JavaScript: -

1. Use var keyword to make data members private.
2. Use setter methods to set the data and getter methods to get that data.

The encapsulation allows us to handle an object using the following properties:

- **Read/Write** - Here, we use setter methods to write the data and getter methods read that data.
- **Read Only** - In this case, we use getter methods only.
- **Write Only** - In this case, we use setter methods only.

You can find the example of encapsulation at link.

Inheritance

A concept called JavaScript inheritance enables us to build new classes off of preexisting ones. It gives the child class the freedom to repurpose the parent class's methods and variables. To build a child class off of a parent class, use the JavaScript extends keyword. It makes it easier for kid classes to inherit all of the traits and behaviours of their parent classes.

Keypoints:

1. It continues to have an IS-A connection.
2. In class declarations or expressions, the extends keyword is utilised.
3. We may get all of the built-in object's properties and behaviour, as well as that of custom classes, by using the extends keyword.
4. Inheritance can also be accomplished via a prototype-based strategy.

You can find the example of inheritance at link.

Polymorphism

A fundamental idea of an object-oriented paradigm that offers a mechanism to carry out a single operation in various forms is called polymorphism. It gives the option of calling the same method on various JavaScript objects. We are free to use any type of data member because JavaScript is not a type-safe language.

You can find the example of polymorphism at link.

Abstraction

An abstraction is a technique for keeping implementation specifics hidden from consumers and simply displaying functionality. In other words, it ignores the extraneous information and just displays what is necessary.

Keypoints:

1. We cannot create an instance of Abstract Class.
2. It reduces the duplication of code.

You can find the example of abstraction at link.