Day 15 - Classes

Classes

JavaScript is an object oriented programming language, meaning everything in JS is an object with properties and methods. We create a class to create an object. A class is essentially an object constructor or ‘blueprint’ for creating objects. Creating an instance of a class creates an object, and the class defines attributes and the behavior of an object, while the object represents the class.

Once we make a class, we can create an object from it whenever we want. This is called instantiation.

An object literal is a singleton. If we want a similar object, we need to write it. However, a class allows us to create many objects, which helps to reduce the amount of code and repetition of code.

Defining a class

To define a class, we need to use the keyword *class*, the name of the class in **CamelCase** and block code(two curly brackets).

```
class ClassName {
  // code goes here
}
```

Class instantiation

AKA creating an object from a class. We need to use the keyword *new* and call the name of the class after.

Creating a person object from our Person class:

```
class Person {
  // code goes here
}
const person = new Person()
console.log(person)

//log:
Person {}
```

Since the class doesn’t have any properties or methods, our object doesn’t either. We can use the class constructor to pass different properties for the class.

Class Constructor

The constructor is a built-in function that allows you to create a blueprint for an object. The constructor function starts with the keyword *constructor* followed by a parenthesis. Inside the parenthesis, we pass the

properties of the object as parameters. We use the *this* keyword to attach the constructor parameters with the class.

```
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
  }
}

const person = new Person()

console.log(person)

//log:
Person {firstName:undefined, lastName:undefined}
```

All keys of the object are undefined. When we instantiate, we should pass the value of the properties:

```
class Person() {
  constructor(firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
  }
}

const person1 = new Person('Chris', 'Guerrero')

console.log(person1)

//log:
Person {firstName: 'Chris', lastName: 'Guerrero'}
```

Creating multiple Person objects:

```
const person2 = new Person('Cinthia', 'Guerrero')
const person3 = new Person('Alexander', 'Guerrero')
```

Creating more properties in the Person class:

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    console.log(this)
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
  }
}

const person1 = new Person('Chris', 'Guerrero', 24, 'United States', 'Phoenix')
```

Default values with constructor

Constructor function properties may have a default value like other regular functions.

```
class Person {
  constructor(
    firstName = 'Chris',
    lastName = 'Guerrero',
    age = 250,
    country = 'USA',
    city = 'Phoenix'
  ) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
  }
}

const person1 = new Person() // it will take the default values
const person2 = new Person('Lidiya', 'Tekle', 28, 'Finland', 'Espoo')
```

Class methods

In a class, we can create class methods. These are JavaScript functions inside the class.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
}
```

Properties with initial value

When creating a class for some properties, we may have an initial value. For example, you're playing a game, so you want the starting score to be zero. We therefore should set the starting score to zero. As for our person class, we may have an initial skill and want to acquire some skill after some time.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
```

```
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
}

const person1 = new Person('Chris', 'Guerrero', 250, 'USA', 'Phoenix')

console.log(person1.score)

console.log(person1.skills)

//log:
0
[]
```

A method could be a regular method, a getter, or a setter.

getter

The get method lets us access value from an object. We write a get method using keyword *get* followed by a function. Instead of accessing properties directly from the object, we use getter to get the value.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
  get getScore() {
    return this.score
  }
  get getSkills() {
    return this.skills
  }
}

const person1 = new Person ('Chris', 'Guerrero', 250, 'USA', 'Arizona')

console.log(person1.getScore) // We don't need parenthesis to call a getter function
console.log(person1.getSkills)

//log:
0
[]
```

setter

Setter methods allow us to modify the value of certain properties. We write a setter method using keyword *set* followed by a function.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
  get getScore() {
    return this.score
  }
  get getSkills() {
    return this.skills
  }
  set setScore(score) {
    this.score += score
  }
  set setSkills(skill) {
    this.skills.push(skill)
  }
}

const person1 = new Person ('Chris', 'Guerrero', 250, 'USA', 'Arizona')

person1.setScore = 1
person1.setSkill = 'HTML'
person1.setSkill = 'CSS'
person1.setSkill = 'JavaScript'

console.log(person1.score)
console.log(person1.skills)

//log:
1
["HTML", "CSS", "JavaScript"]
```

Don’t confuse regular methods with getters and setters. Lets create a regular method “getPersonInfo()”:

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
  get getScore() {
    return this.score
  }
  get getSkills() {
```

```
    return this.skills
  }
  set setScore(score) {
    this.score += score
  }
  set setSkills(skill) {
    this.skills.push(skill)
  }
  getPersonInfo() {
    let fullName = this.getFullName()
    let skills =
      this.skills.length > 0 &&
      this.skills.slice(0, this.skills.length - 1).join(', ') +
        ` and ${this.skills[this.skills.length - 1]}`
    let formattedSkills = skills ? `He knows ${skills}` : ''

    let info = `${fullName} is ${this.age}. He lives ${this.city}, ${this.country}. ${formattedSkills}`
    return info
  }
}

const person1 = new Person ('Chris', 'Guerrero', 250, 'USA', 'Arizona')

person1.setScore = 1
person1.setSkill = 'HTML'
person1.setSkill = 'CSS'
person1.setSkill = 'JavaScript'

console.log(person1.score)
console.log(person1.skills)
console.log(person1.getPersonInfo())
```

Static method

The static keyword defines a static method for a class. Static methods aren't called on instances of the class. Instead, they're called on the class itself. These're often utility functions, such as functions to create or clone objects. An example of a static method is Date.now(). The now method is called directly from the class. Another example is Object.assign(), which is used to copy an object.

```
class Person {
  constructor(firstName, lastName, age, country, city) {
    this.firstName = firstName
    this.lastName = lastName
    this.age = age
    this.country = country
    this.city = city
    this.score = 0
    this.skills = []
  }
  getFullName() {
    const fullName = this.firstName + ' ' + this.lastName
    return fullName
  }
  get getScore() {
    return this.score
  }
  get getSkills() {
    return this.skills
  }
  set setScore(score) {
    this.score += score
```

```
  }
  set setSkill(skill) {
    this.skills.push(skill)
  }
  getPersonInfo() {
    let fullName = this.getFullName()
    let skills =
      this.skills.length > 0 &&
      this.skills.slice(0, this.skills.length - 1).join(', ') +
        ` and ${this.skills[this.skills.length - 1]}`

    let formattedSkills = skills ? `He knows ${skills}` : ''

    let info = `${fullName} is ${this.age}. He lives ${this.city}, ${this.country}. ${formattedSkills}`
    return info
  }
  static favoriteSkill() {
    const skills = ['HTML', 'CSS', 'JS', 'React', 'Python', 'Node']
    const index = Math.floor(Math.random() * skills.length)
    return skills[index]
  }
  static showDateTime() {
    let now = new Date()
    let year = now.getFullYear()
    let month = now.getMonth() + 1
    let date = now.getDate()
    let hours = now.getHours()
    let minutes = now.getMinutes()
    if (hours < 10) {
      hours = '0' + hours
    }
    if (minutes < 10) {
      minutes = '0' + minutes
    }

    let dateMonthYear = date + '.' + month + '.' + year
    let time = hours + ':' + minutes
    let fullTime = dateMonthYear + ' ' + time
    return fullTime
  }
}

console.log(Person.favoriteSkill())
console.log(Person.showDateTime())
```

Inheritance

Inheritance allows us to access all the properties and methods of the parent class, which reduces repetition of code. In the example above, we have a Person parent class and we will create children from it. Our children class could be student, teacher, etc.

```
// syntax
class ChildClassName extends {
  // code goes here
}
```

Creating a student child class from Person parent class:

```
class Student extends Person {
  saySomething() {
    console.log('I am a child of the person class')
  }
}

const s1 = new Student('Christopher', 'Guerrero', 250, 'USA', 'Phoenix')

console.log(s1)
console.log(s1.saySomething())
console.log(s1.getFullName())
console.log(s1.getPersonInfo())
```

Overriding methods

As seen above, we can access all the methods in the Person class and use it in the Student child class. We can customize the parent methods, such as adding additional properties to a child class. If we want to customize a parent method or add extra properties, we need to use the constructor function of the child class. Inside the constructor function, we call the super() function to access all the properties from the parent class. The Person class didn't have gender, so we can add the gender property to the child class, Student. If the same method name is used in the child class, the parent method will be overwritten.

```
class Student extends Person {
  constructor(firstName, lastName, age, country, city, gender) {
    super(firstName, lastName, age, country, city)
    this.gender = gender
  }

  saySomething() {
    console.log('I am a child of the person class')
  }
  getPersonInfo() {
    let fullName = this.getFullName()
    let skills =
      this.skills.length > 0 &&
      this.skills.slice(0, this.skills.length - 1).join(', ') +
        ` and ${this.skills[this.skills.length - 1]}`

    let formattedSkills = skills ? `He knows ${skills}` : ''
    let pronoun = this.gender == 'Male' ? 'He' : 'She'

    let info = `${fullName} is ${this.age}. ${pronoun} lives in ${this.city}, ${this.country}. ${formattedSkills}`
    return info
  }
}

const s1 = new Student(
  'Asabeneh',
  'Yetayeh',
  250,
  'Finland',
  'Helsinki',
  'Male'
)
const s2 = new Student('Lidiya', 'Tekle', 28, 'Finland', 'Helsinki', 'Female')
s1.setScore = 1
s1.setSkill = 'HTML'
s1.setSkill = 'CSS'
s1.setSkill = 'JavaScript'
```

```
s2.setScore = 1
s2.setSkill = 'Planning'
s2.setSkill = 'Managing'
s2.setSkill = 'Organizing'

console.log(s1)

console.log(s1.saySomething())
console.log(s1.getFullName())
console.log(s1.getPersonInfo())

console.log(s2.saySomething())
console.log(s2.getFullName())
console.log(s2.getPersonInfo())
```

Above, the getPersonInfo method has been overwritten and identifies if the person is male or female.