모듈

2023.06.13.

Note.
모듈은 HTTP(s) 프로토콜에서만 작동합니다.
file:// 프로토콜을 통해 열린 웹 페이지는 import/export를 사용할 수 없습니다.

Visual Studio Code에서 실행 방법
1. 확장 : Live Server 설치 후 Visual Studio Code 재시작
2. html 파일에서 마우스 오른쪽 > Open With Live Server

1. 모듈
JavaScript 모듈을 사용하면 코드를 별도의 파일로 나눌 수 있습니다.
이렇게 하면 코드 기반을 유지하기가 더 쉬워집니다.
import 문을 사용하여 외부 파일에서 모듈을 가져옵니다.
또한 모듈은 <script> 태그에 type="module"에 의존합니다.

구문

```
<script type="module">
  import 객체명 from "객체가 있는 소스파일(js)의 경로";
</script>
```

예시

```
<script type="module">
  import message from "./message.js";
</script>
```

2. Export
함수 또는 변수가 있는 모듈은 모든 외부 파일에 저장할 수 있습니다.
내보내기에는 명명된 내보내기(Named Exports) 와 기본 내보내기(Default Export) 의 두 가지 유형이 있습니다.

2.1 명명된 내보내기
person.js라는 이름의 파일을 생성하여 내보낼 항목으로 채우도록 합니다.
명명된 내보내기는 개별적인 인라인 방식과 하단에 한 번에 내보내는 방식으로 내보내기를 만들 수 있습니다.

2.1.1 개별적 인라인 방식(person.js)

```
export const name = "Jesse";
export const age = 40;
```

2.1.2 하단에서 한 번에 내보내는 방식(person.js)

```
const name = "Jesse";
const age = 40;

export {name, age};
```

2.2 기본 내보내기

message.js 라는 다른 파일을 생성하여 기본 내보내기를 보여주는 데 사용합니다.
파일에는 기본 내보내기를 하나만 사용할 수 있습니다.

```
const message = function() {
  const name = "Jesse";
  const age = 40;
  return name + ' is ' + age + 'years old.';
};

export default message;
```

3. Import

명명된 내보내기인지 또는 기본 내보내기인지에 따라 두 가지 방법으로 모듈을 파일로 가져올 수 있습니다.
명명된 내보내기는 중괄호를 사용하여 구성됩니다. 기본 내보내기는 그렇지 않습니다.

3.1 명명된 내보내기에서 가져오기

person.js 파일에서 명명된 내보내기 가져오기:

```
import { name, age } from "./person.js";
```

3.2 기본 내보내기에서 가져오기

message.js 파일에서 기본 내보내기를 가져옵니다.

```
import message from "./message.js";
```

ES6으로도 알려진 ECMAScript 2015는 JavaScript 클래스를 도입했습니다.
JavaScript 클래스는 JavaScript 객체용 템플릿입니다.

1. 자바스크립트 클래스 구문

키워드 class를 사용하여 클래스를 만듭니다.
항상 이름이 지정된 constructor() 메소드를 추가하십시오 .

구문

```
class 클래스명 {
  constructor() {
    this.속성명 = 값;
    ...
  }
}
```

예시

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
}
```

위의 예에서는 "Car"라는 클래스를 만듭니다.
이 클래스에는 "name"과 "year"라는 두 개의 초기 속성이 있습니다.

2. 클래스 사용

클래스가 있으면 클래스를 사용하여 개체(object)를 만들 수 있습니다.

구문

```
const 객체명 = new 클래스명(인수, ...);
```

예시

```
const myCar1 = new Car("Ford", 2014);
const myCar2 = new Car("Audi", 2019);
```

위의 예에서는 Car 클래스를 사용하여 두 개의 Car 객체를 만듭니다 .
생성자 메서드는 새 객체가 생성될 때 자동으로 호출됩니다.

3. 생성자 메서드

생성자 메서드는 특별한 메서드입니다:
① 정확한 이름 "constructor"가 있어야 합니다.
② 새로운 객체가 생성되면 자동으로 실행됩니다.
③ 개체 속성을 초기화하는 데 사용됩니다.
생성자 메서드를 정의하지 않으면 JavaScript는 빈 생성자 메서드를 추가합니다.

4. Class Methods

클래스 메서드는 개체(Object) 메서드와 동일한 구문으로 생성됩니다.

키워드 class를 사용하여 클래스를 만듭니다.
항상 constructor() 메서드를 추가하십시오.
그런 다음 여러 메서드를 추가합니다.

구문

```
class 클래스명 {
  constructor() { ... }
  메서드_1() { ... }
  메서드_2() { ... }
  메서드_3() { ... }
}
```

Car age를 반환하는 "age"라는 클래스 메서드를 만듭니다.

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age() {
    const date = new Date();
    return date.getFullYear() - this.year;
  }
}

const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML = "My car is " + myCar.age() + " years old.";
```

클래스 메소드에 매개변수를 보낼 수 있습니다.

```
class Car {
  constructor(name, year) {
    this.name = name;
    this.year = year;
  }
  age(x) {
    return x - this.year;
  }
}

const date = new Date();
let year = date.getFullYear();

const myCar = new Car("Ford", 2014);
document.getElementById("demo").innerHTML = "My car is " + myCar.age(year) + " years old.";
```

02. 상속

2023.06.12.

1. 클래스 상속

클래스 상속을 만들려면 extends 키워드를 사용하십시오.
클래스 상속으로 생성된 클래스는 다른 클래스의 모든 메서드를 상속합니다.

구문

```
class 클래스명 {
  super(인수, ...);
  ...
}
```

메서드 super()는 부모 클래스를 참조합니다.
생성자 메서드에서 메서드를 호출하여 super()부모의 생성자 메서드를 호출하고 부모의 속성 및 메서드에 대한 액세스 권한을 얻습니다.
상속은 코드 재사용에 유용합니다. 새 클래스를 만들 때 기존 클래스의 속성과 메서드를 재사용합니다.

예시
"Car" 클래스에서 메서드를 상속할 "Model"이라는 클래스를 만듭니다.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

let myCar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = myCar.show();
```

2. 게터와 세터

클래스를 사용하면 getter 및 setter를 사용할 수도 있습니다.
특히 값을 반환하기 전에 또는 설정하기 전에 값으로 특별한 작업을 수행하려는 경우 속성에 getter 및 setter를 사용하는 것이 현명할 수 있습니다.
클래스에 getter 및 setter를 추가하려면 get및 set키워드를 사용합니다.

구문

```
class 클래스명 {
  construcotr() { ... }
  get 게터명() {
    ...
    return this.속성명;
  }
  set 세터명(매개변수) {
    ...
    this.속성명 = 값;
  }
}

const 객체명 = new 클래스명();

객체명.세터명 = 값;
conslole.log(객체명.게터명);
```

예시
"carname" 속성에 대한 getter 및 setter를 만듭니다.

```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  get cnam() {
    return this.carname;
  }
  set cnam(x) {
    this.carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.cnam;
```

참고: getter가 메서드인 경우에도 속성 값을 가져오려는 경우 괄호를 사용하지 않습니다.

getter/setter 메서드의 이름은 속성 이름(이 경우 carname)과 같을 수 없습니다.
많은 프로그래머는 getter/setter를 실제 속성과 구분하기 위해 속성 이름 앞에 밑줄(_) 문자를 사용합니다.

예시
밑줄 문자를 사용하여 getter/setter를 실제 속성과 구분할 수 있습니다.

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = myCar.carname;
```

setter 를 사용하려면 괄호 없이 속성 값을 설정할 때와 동일한 구문을 사용합니다.

예시
setter를 사용하여 carname을 "Volvo"로 변경합니다.

```
class Car {
  constructor(brand) {
    this._carname = brand;
  }
  get carname() {
    return this._carname;
  }
  set carname(x) {
    this._carname = x;
  }
}

const myCar = new Car("Ford");
myCar.carname = "Volvo";
document.getElementById("demo").innerHTML = myCar.carname;
```

3. Hoisting(게양)
함수 및 기타 JavaScript 선언과 달리 클래스 선언은 호이스팅되지 않습니다.
즉, 클래스를 사용하려면 먼저 클래스를 선언해야 합니다.

예시

```
//myCar = new Car("Ford") 아직 클래스를 사용할 수 없습니다. 오류가 발생합니다.

class Car {
  constructor(brand) {
    this.carname = brand;
  }
}

const myCar = new Car("Ford") //이제 클래스를 사용할 수 있습니다.
```

참고: 함수 와 같은 다른 선언의 경우 JavaScript 선언의 기본 동작이 호이스팅(선언을 맨 위로 이동)하

기 때문에 선언하기 전에 사용하려고 하면 오류가 발생하지 않습니다.

03. Static 메서드

2023.06.12.

static 클래스 메서드는 클래스 자체에 정의됩니다.
static 메서드는 생성된 객체(object)에서 호출할 수 없으며 클래스명을 통해서만 호출할 수 있습니다.

예시

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello() {
    return "Hello!!";
  }
}

const myCar = new Car("Ford");

document.getElementById("demo").innerHTML = Car.hello();
//Car 클래스에서 hello() 메서드를 호출할 수 있습니다.

//document.getElementById("demo").innerHTML = myCar.hello();
//myCar 객체에서는 호출할 수 없습니다. 에러가 발생 됩니다.
```

static 메서드 내에서 myCar 개체를 사용하려면 매개 변수로 보낼 수 있습니다.
예시

```
class Car {
  constructor(name) {
    this.name = name;
  }
  static hello(x) {
    return "Hello " + x.name;
  }
}
const myCar = new Car("Ford");
document.getElementById("demo").innerHTML = Car.hello(myCar);
```