자바스크립트의 시작

1. 웹과 Javascript

1) WEB과 Javascript, HTML

- 웹이 처음 등장했을 때, 사람들은 HTML을 사용해서 정보를 주고 받았습니다. 하지만, HTML은 정적입니다. 한 번 출력되면 그 모양이 바뀌지 않습니다. 사용자와 동적으로 사용하는 웹을 만들기 위해 자바스크립트가 등장했습니다. HTML을 이용해 웹페이지를 만들고, 자바스크립트를 이용해 사용자와 상호작용할 수 있도록 추가한다. 즉, 정적인 정보인 HTML을 자바스크립트가 동적으로 만들어 준다.

2) Javascript의 역할

- 가장 중요한 역할은 사용자와 상호작용할 수 있게 하는 것입니다.

3) HTML과 JS의 만남 : script 태그

- HTML과 Javascript : 자바스크립트는 HTML 위에서 동작하는 언어다. 두 언어를 하나로 합칠 때, script 태그가 필요하다.
- script 태그 : HTML의 태그 중 하나인 script 태그 안에는 자바스크립트 코드를 쓸 수 있다.

```
<body>
  <script>
    document.write('hello, world!');
  </script>
</body>
```

위와 같이 <body> 태그 안에 넣어서 웹페이지를 띄워보면 페이지에 hello,world! 라는 글자가 뜨는 것을 볼 수 있습니다. 하지만 이러한 기능은 다음과 같이 HTML만 사용해서 구현할 수도 있습니다.

```
<body>
  hello, world!
</body>
```

출력 결과는 같다. 이 둘의 차이점은? 자바스크립트로 쓴 코드는 동적이라는 점! 만약, hello, world! 대신 1+1을 출력한다고 가정할 때 HTML은 정적이기에 문자 그대로 1+1을 출력하지만, 자바스크립트는 동적으로 이를 계산해 2로 출력할 수 있다.

4) HTML과 JS의 만남 : 이벤트

- Javascript와 사용자의 상호작용, 이벤트(Event) : alert('hi') 코드를 통해 alert 창을 만들 수 있습니다. 그렇다면 이제, 사용자가 어떤 버튼을 눌렀을 때 이 alert 창이 뜨도록 만드는 법은 먼저 빈 화면에 버튼을 만들고, 이 버튼을 눌렀을 때 어떤 동작이 실행되도록 하는지 지정하는 속성인 onclick에 이 javascript 코드를 넣어봅시다.

```
<input type="button" value="hi" onclick="alert('hi')">
```

버튼을 눌렀을 때 경고창이 뜨게 된다.

- Onclick 속성 : HTML 태그 안에서 onclick 속성은 javascript 코드를 가지게 됩니다. 그리고 onclick이 포함된 태그가 클릭되었을 때 이 javascript 코드에 따라서 웹 브라우저가 동작됩니다. 즉, 위의 코드에서 웹 브라우저는 alert('hi') 라는 코드를 기억하고 있다가, 사용자가 클릭하면 이를 실행해주는 것입니다. 이
- 이벤트(event) : 웹 브라우저에서 일어나는 사건으로, 이벤트의 종류는 onclik, onchang 등이 있다. onclik은 사용자가 어떤 것을 클릭하는 사건을 의미한다. onchange는 입력창에서 사용자가 키보드를 이용해 무언가 입력하면, 그에 따라 내용이 바뀌는 사건을 의미하는 것입니다. 이때, 입력창에 입력한 후 바

깥쪽을 클릭했을 때를 기준으로 그 전과 내용이 바뀌었는지 확인한다는 점을 알아둬야 한다. 이 외에도 총 10~20가지 정도의 이벤트가 존재합니다. 이를 이용해서 사용자와 상호작용하는 웹 사이트를 만들 수 있게 됩니다.

https://www.w3schools.com/js/js_events.asp

5) HTML과 JS의 만남 : 콘솔

- 콘솔(Console) : 웹 브라우저에서 오른쪽 버튼 > 검사 를 누르면 뜨는 창을 잘 살펴보면 Console이라는 이름의 탭이 보일 겁니다. 이 콘솔을 이용하면 파일을 만들지 않고도 바로 Javascript 코드를 실행할 수 있습니다. 어떤 문자열(Hello, World!)의 길이를 알고 싶다고 해 봅시다. 이때 바로 이 콘솔 창에서 다음과 같이 치고 엔터를 누르면 문자열의 길이를 현재 페이지에서 바로 경고창으로 출력할 수 있습니다.

```
alert('Hello, World!'.length)
```

즉, 콘솔을 통해서 입력된 코드는 현재 페이지에서 즉석으로 실행되는 것입니다.

6) 데이터 타입(문자열과 숫자)

- 숫자(Number) : 예를 들어 1+1을 실행시키면 2가 출력될 겁니다. 숫자 데이터 타입의 가장 중요한 점은 연산이 가능하다는 것입니다. 즉, 1+1에서는 +라는 연산이 왼쪽에 있는 값과 오른쪽에 있는 값을 더해서 하나의 값을 만들어내게 됩니다. 그렇기에 +를 이항 연산자라고 부릅니다. 계산을 하기 때문에 산술 연산자라고 부르기도 합니다. 여러분이 흔히 아는 사칙 연산 (+, -, *, /)는 모두 산술 연산자에 속합니다. 즉, 사칙 연산을 이용해서 숫자 데이터 타입을 계산할 수 있는 것입니다.
- 문자열(String) :

문자열은 따옴표로 시작해서 따옴표로 끝나게 됩니다. 작은 따옴표로 시작하면 작은 따옴표로 끝내고, 큰 따옴표로 시작하면 큰 따옴표로 끝내면 됩니다. 문자열에서 흔히 사용하는 연산으로는 length가 있습니다. 문자열 뒤에 .을 붙이고 length를 입력하면 그 문자열의 길이를 알려주게 됩니다. 그 외에도 문자를 처리하는 데 다양한 명령어를 사용할 수 있습니다. 예를 들면 str.toUpperCase()를 사용하면 str 안에 들어있는 모든 문자열의 문자들이 대문자로 바뀌게 됩니다. str.indexOf('hi')를 하면 str 안에 있는 hi 라는 문자열을 찾아서 그 부분이 앞에서부터 몇 번째 문자인지 알려주게 됩니다.

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/String

https://developer.mozilla.org/ko/docs/Web/JavaScript/Data_structures

7) 변수와 대입 연산자

- 변수 : 그대로 바뀔 수 있는 수, 변할 수 있는 수

```
x=1;
y=1;
x+y;
```

x+y의 결과값은 2이다. 하지만, 코드를 이어서 실행하면

```
x=1000;
x+y;
```

같은 x+y이지만, x의 값이 1000으로 바뀌었기 때문에 결과는 1001이 될 겁니다. x처럼 값이 바뀔 수 있는 것을 변수라고 합니다. 이때 =는 대입 연산자라고 부릅니다. 왼쪽에 있는 변수에 오른쪽에 있는 숫자를 대입한다는 의미입니다.

```
1=2;
```

실행하면 에러가 난다. 1은 변수가 아니기에 1은 언제나 1이다. 이렇게 바뀌지 않는 수를 상수라고 합니다.

- 변수의 필요성 : 변수를 이용해서 여러군데 흩어져 있는 값들을 한 번에 바꿀 수 있다는 것입니다. 예

를 들어 한 문자열 안에서 같은 "hello"라는 단어가 여러 번 나올 때, 이를 모두 "hi"로 바꾸고 싶다고 해 봅시다. 이때 모든 "hello"들이 변수로 나타나 있다면, 변수만 바꿔주면 한 번에 모든 hello들을 hi로 바꿀 수 있다는 것입니다. 변수를 쓸 때 한 가지 알아두시면 좋은 점은 다음 코드와 같이 변수 이름 앞에 var을 붙이면 좋다는 점입니다. 이때 var은 변수의 영어 단어인 variable의 약자입니다.

```
var word = "hello"
```

8) 웹 브라우저 제어

- 웹 브라우저 제어 :

```
<body>
<body style="background-color: black; color: white;">
```

원래 웹페이지에서는 첫 줄과 같이 <body> 태그만 존재했습니다. 하지만 원래의 <body> 태그에서 두 번째 줄과 같이 style을 지정해주면, 배경색은 검정색으로, 글자 색은 하얀색으로 바뀌게 됩니다. 이때 style 속성에 들어있는 간단한 코드를 css라고 부릅니다. css는 디자인을 위한 언어이고, HTML, Javascript와는 완전히 다른 언어입니다. 하지만 HTML은 한 번 표시되면 바뀌지 않는 정적인 언어입니다. 즉 <body> 태그가 만들어지면, 저 style 속성값을 바꿀 수 없다는 이야기입니다.

9) CSS 기초(style 속성)

- CSS의 기본 문법 : CSS를 이용하면 웹 페이지에 있는 요소들의 디자인을 바꿀 수 있습니다. 이를 위해 style 속성을 사용하면 됩니다. 이 style 속성 안에는 CSS가 들어가게 됩니다.

<h1>Javascript<h1> 이렇게 하면 Javascript 라는 글자가 나타나게 됩니다. 글자의 색을 blue로 바꾸고 싶다면 <h1 style="color: blue">Javascript<h1>과 같이 써주면 됩니다. 이때 color: blue라는 코드가 CSS입니다. CSS를 이용하면 다양한 종류의 디자인 요소를 바꿀 수 있습니다. 예를 들어 color 대신 background-color을 쓰면 배경 색이 바뀌는 것을 볼 수 있습니다.

https://www.w3schools.com/cssref/pr_background-color.asp

10) CSS 기초(style 태그)

- Span과 Div : 문단의 특정 부분에만 스타일을 주어서 강조하고 싶을 때 사용한다. 둘 모두 어떠한 특정한 기능이 있는 태그가 아니고, CSS나 Javascript 코드를 삽입하기 위해서 존재하는 태그입니다. div 태그는 화면 전체를 사용하기 때문에 줄바꿈이 되고, span은 줄바꿈이 되지 않습니다. 이러한 div와 span 태그 안에 style 속성을 부여하면 이 태그로 감싸진 부분에만 디자인이 적용되게 됩니다. 하지만 이렇게 모든 부분을 div나 span 태그로 감싸려고 한다면 이를 쓰기도 힘들고, 수정하기는 더욱 힘들 것입니다.

- Class : <head> 태그 안에 <style>이라는 새로운 태그를 만들어 줍니다. 이 style 태그 안에는 CSS 코드가 들어가게 됩니다. 그리고 js라는 class를 생성해 봅시다.

```
<head>
  <style>
    .js {
        font-weight: bold;
    }
  </style>
</head>
```

이때 js 앞에 찍혀있는 점은 js라는 이름이 class 이름이라는 것을 나타냅니다. 이렇게 class를 만들었으니, 이를 HTML 코드의 곳곳에 적용시키면 됩니다. 예를 들어 어떤 문장에서 Javascript라는 단어에만 이 class를 적용시켜서 볼드체로 만들고 싶다면 Javascript is wonderful! 이렇게 하면 이제 js라는 class를 가진 모든 태그를 한 번에 바꿀 수 있습니다. style 태그 안에 있는 .js만 수정

하면 모든 부분을 한 번에 바꿀 수 있는 것이다.

11) CSS 기초(선택자)

- 선택자 : class와 id가 존재한다. class에 대한 스타일을 지정할 때 <head> 태그 안에 넣어주었다. id는 class와 달리, .대신에 #을 붙여야 한다는 차이점이 있다.

```
<style>
  #first {
    color: green;
  }
</style>
```

- class와 id의 차이점 :class라는 단어는 그룹을 의미하고, id는 특정한 것을 식별한다는 의미입니다. class 선택자는 같은 스타일이 지정될 그룹을 의미하는 것이고, id는 특정한 태그에만 지정하고 싶은 스타일을 나타내는 것입니다. 그래서 class는 중복될 수 있지만, id는 한 페이지에서는 딱 한 번만 쓰이게 되는 것입니다. 즉, class 선택자가 id 선택자에 비해서 더 포괄적입니다. class 선택자를 이용하면 광범위한 효과를 줄 수 있고, id 선택자를 이용하면 예외적으로 디자인을 바꿀 수 있는 것입니다. 그렇기에 class 위에 id를 얹어서 구현하는 것이 효율적입니다.

- 선택자의 우선순위 : 마지막으로 id 선택자와 class 선택자 외에 스타일을 나타내는 한 가지 방법을 더 알아봅시다. 만약 앞에 .과 # 모두 지정하지 않으면 그것은 태그 이름을 의미합니다. 즉 아래와 같은 코드를 사용하면 이 페이지에서 span이라는 이름의 태그는 모두 디자인이 바뀌는 것입니다.

```
<style>
  span {
    color: blue;
  }
</style>
```

Javascript 이 코드의 색은? 이 태그는 span이기 때문에 파란색이 될 수도 있지만, id가 first이기 때문에 초록색이 될 수도 있고, class가 js이기 때문에 빨간색이 될 수도 있겠지만, 답은 초록색입니다. id의 우선순위가 가장 높은 것입니다. 만약 id를 지우게 되면 class의 속성이었던 빨간색이 나타나게 됩니다. id와 class를 모두 지워야지만 span의 스타일인 파란색이 나타납니다. 즉, id > class > 태그 순서로 우선순위를 가지게 됩니다.

12) 제어할 태그 선택하기

- 제어할 태그 선택하기 : 이벤트가 일어났을 때, 어떤 태그에 스타일이 지정될지 선택하는 작업이 필요합니다.

- querySelector : 함수를 사용하면 이러한 선택자를 이용해서 원하는 태그를 선택할 수 있습니다.

```
documnet.querySelector("body")
```

이렇게 하면 페이지 내에서 body라는 이름의 태그를 모두 선택하는 것이죠. 만약 js라는 class를 가진 태그를 선택하고 싶다면 따옴표 사이에 .js를 쓰면 되고, first라는 id를 가진 태그를 선택하고 싶다면 #first라고 쓰면 됩니다.

```
documnet.querySelector("body").style.backgroundColor = 'black';
```

body 태그를 모두 고른 뒤, 여기에 스타일을 적용하기 위해서 style이라고 써 주고, 여러 스타일 중에서도 배경색을 지정하기 위해서 backgroundColor라고 써 준 것입니다. 이렇게 완성한 Javascript 코드를 이벤트가 일어날 때마다 실행하면 되는 것입니다. 예를 들어서, 버튼을 클릭할 때 이러한 스타일 변화가 일어나도록 만들려면 <input type="button" value="night" onclick="documnet.querySelector('body').style.backgroundColor = 'black';">

https://developer.mozilla.org/ko/docs/Web/API/Document/querySelector

2. Javascript 제어문

1) 프로그램, 프로그래밍, 프로그래머

- Javascript란 무엇인가? HTML과 Javascript는 모두 컴퓨터 언어입니다. 하지만 HTML과는 달리 Javascript는 컴퓨터 프로그래밍 언어이기도 하다.
- 프로그램, 프로그래머, 프로그래밍 : 프로그램에는 순서라는 의미가 있습니다. 프로그래밍은 이러한 순서를 만드는 행위를 말하죠. 프로그래머는 이러한 순서를 만드는 일을 하는 사람을 의미합니다. 이러한 용어는 소프트웨어 분야뿐 아니라 다양한 분야에서도 사용되는 말입니다. 컴퓨터를 사용할 때는 다양한 기능을 순서대로 사용하게 됩니다. 그리고 보통은 이러한 기능이 반복적으로 이용됩니다. 컴퓨터 프로그래밍 언어란 시간의 순서에 따라서 실행되어야 할 기능을 글로 적어두는 방식을 의미합니다. 작업이 필요할 때마다 적어둔 글을 컴퓨터가 실행하도록 건네주는 것이다.
- HTML과 Javascript의 비교 : HTML로 만든 웹페이지는 시간의 순서에 따라 실행되지 않고, 한 번 만들어지면 바뀌지 않습니다. 때문에 HTML은 컴퓨터 프로그래밍 언어가 아닌 것입니다. 반면에 Javascript는 사용자와 상호작용하고, 이를 위해서 시간에 따라 여러 기능이 실행되어야 하기 때문에 프로그래밍이라는 형태를 띄게 됩니다. 따라서 Javascript는 컴퓨터 프로그래밍 언어라고 부를 수 있는 것입니다. 그리고 더 나아가서 시간에 따라 코드가 실행되는 것 외에도, 조건에 따라 다른 코드가 실행되도록 하거나, 같은 코드가 반복적으로 실행할 수 있는 방법도 고안하게 된 것입니다.

2) 조건문 예고

- 조건문 : 프로그램이 조건에 따라서 다른 기능들이 다른 순서에 따라서 실행되도록 만들어주는 것입니다. 조건문은 단순하고 반복적인 업무뿐 아니라 복잡한 업무까지도 컴퓨터가 다룰 수 있도록 해 줍니다. 버튼을 두 개 만들 필요 없이 하나로 구현할 수 있게 될 겁니다. 이러한 버튼을 토글이라고 부릅니다.

3) 비교연산자와 Boolean

- 비교 연산자 === : 동등 비교연산자(===) 왼쪽과 오른쪽이 같은지 판단하는 것이다. 이항 연산자로 좌항과 우항을 결합해서 그 관계에 따라 하나의 결과, 즉 True, False인지 만들어냅니다.
- Boolean : 비교 연산자를 사용하면 결과로 True 또는 False가 만들어집니다. 이때 이 True와 False를 보고 Boolean 데이터 타입이라고 부릅니다.
- 비교 연산자 <, > : 두 값의 크기를 서로 비교하는 연산자가 있습니다. HTML에서는 이 꺽쇠 기호(<,>)가 태그를 나타낼 때 쓰이기 때문에 혼란을 줄 수 있기 때문이죠. <가 <를 의미하는 것이고, >가 >를 의미합니다. Javascript에서는 그대로 <, >로 나타내도 괜찮습니다.

4) 조건문

- 조건문

```
<script>
  document.write('1')
  if(true) {
    document.write('2')
  }
  else {
    document.write('3')
  }
```

```
</script>
 -> 1 2
<script>
  document.write('1')
  if(false) {
    document.write('2')
  }
  else {
    document.write('3')
  }
</script>
 -> 1 3
```

if문의 괄호 안에는 boolean 데이터 타입이 옵니다. 그리고 만약 그 boolean이 true이면 첫 번째 중괄호에 있는 코드가 실행되고, else의 중괄호에 있는 코드는 무시됩니다. 반대로 그 boolean이 false이면 else의 중괄호에 있는 코드가 실행되고 첫 번째 중괄호에 있는 코드가 무시되어 1 3이 출력됩니다. 괄호 안에 있는 boolean이 고정되어 있기 때문에 항상 같은 결과가 출력되게 됩니다. 하지만, 이 괄호 안에 있는 boolean이 상황에 따라서 true/false가 바뀌도록 만들어주면, 상황에 따라서 다른 코드가 실행되도록 만들어줄 수 있습니다.

5) 조건문의 활용

- 조건문을 활용한 토글 만들기

```
<input id="night_day" type="button" value="night">
```

이 버튼의 value 값을 기준으로, 이 value가 night면 day 버전으로 바뀌는 코드를, day면 night 버전으로 바뀌는 코드를 실행하도록 프로그램을 만들겁니다.

```
if(???) {
  document.querySelector('body').style.backgroundColor = 'black';
  document.querySelector('body').style.color = 'white';
}
else {
  document.querySelector('body').style.backgroundColor = 'white';
  document.querySelector('body').style.color = 'black';
}
```

??? 속에는 어떤 코드가 들어가야 할까요? 우리는 지금부터 document에서 id가 night_day인 버튼을 찾아서, 그 버튼의 value가 "night"인지 아닌지를 찾아내야 합니다. 이를 위해서 다음과 같은 코드를 사용할 수 있습니다.

```
document.querySelector('#night_day').value
```

현재 페이지에서 querySelector을 사용해서 id가 night_day인 태그를 찾기 위해서 id를 나타내는 #을 붙여줍니다. 그리고 찾아낸 태그의 value를 알기 위해서 .value를 써 줍니다. 이렇게 가져온 value 값을 night와 비교하면

```
if(document.querySelector('#night_day').value === 'night') {
  document.querySelector('body').style.backgroundColor = 'black';
  document.querySelector('body').style.color = 'white';
}
else {
```

```
  document.querySelector('body').style.backgroundColor = 'white';
  document.querySelector('body').style.color = 'black';
}
```

하지만 이 코드에는 결정적인 문제가 있습니다. 바로 버튼이 눌릴 때마다 value가 바뀌지 않는 것이죠. 그렇기 때문에 항상 원래 설정된 value인 night에 해당하는 코드, 즉 첫 번째 중괄호에 있는 코드만 실행되게 됩니다. 이를 해결하기 위해서는 코드가 실행될 때마다 value를 바꿔주는 코드도 추가해주면 됩니다.

```
if(document.querySelector('#night_day').value === 'night') {
  document.querySelector('body').style.backgroundColor = 'black';
  document.querySelector('body').style.color = 'white';
  document.querySelector('#night_day').value = 'day';
}
else {
  document.querySelector('body').style.backgroundColor = 'white';
  document.querySelector('body').style.color = 'black';
  document.querySelector('#night_day').value = 'night';
}
```

6) 리팩토링(중복의 제거)

- 리팩토링-this 사용하기 : 리팩토링이란 비효율적인 코드를 효율적으로 만들어서 가독성을 높이고 유지보수가 쉽도록 만드는 것입니다. 코드의 기능적인 면에서는 변화가 없도록 말이다.

자기가 속한 버튼을 찾기 위해서 document.querySelector('#night_day') 라는 코드를 사용했습니다. 하지만 만약 이 코드를 복사해서 하나의 버튼을 더 만들게 된다면 id는 하나의 태그에만 적용될 수 있으므로 새롭게 만들어진 버튼에는 새로운 id값을 적용해줘야겠죠. 예를 들어서 새롭게 만든 버튼의 id를 #night_day2라고 만들었다고 해 봅시다. 그렇게 되면 이 코드에서 모든 #night_day를 #night_day2로 직접 바꿔주는 작업을 진행해야 합니다. 무척 비효율적이다. 그래서 Javascript에는 자기 자신을 가리키기 위한 this라는 키워드가 있습니다. document.querySelector('#night_day') 대신 this를 써도 되는 것이다.

```
if(this.value === 'night') {
  document.querySelector('body').style.backgroundColor = 'black';
  document.querySelector('body').style.color = 'white';
  this.value = 'day';
}
else {
  document.querySelector('body').style.backgroundColor = 'white';
  document.querySelector('body').style.color = 'black';
  this.value = 'night';
}
```

코드가 훨씬 간결해진 것을 볼 수 있습니다. 그 뿐만 아니라, 이 코드는 몇 번을 복사해서 붙여넣더라도 따로 추가적인 수정 없이 계속 사용할 수 있습니다.

- 리팩토링-중복 제거하기 : 두 번째로 리팩토링 해 볼 부분은 바로 document.querySelector('body') 입니다. 이 부분이 무려 4번이나 등장하고 있죠. 코딩을 할 때에는 중복을 없애주는 것이 중요하다.

```
var target = document.querySelector('body');
if(this.value === 'night') {
  target.style.backgroundColor = 'black';
```

```
  target.style.color = 'white';
  this.value = 'day';
}
else {
  target.style.backgroundColor = 'white';
  target.style.color = 'black';
  this.value = 'night';
}
```

이렇게 하면 target이라는 변수를 만들어서 거기에 body 태그를 찾아서 넣고, 이 target 변수만 간단하게 사용해서 코드의 길이를 줄일 수 있습니다. 그 뿐만 아니라 첫 번째 줄만 수정해주면, target이 쓰이는 네 줄을 모두 바꿀 수 있다는 장점도 가지고 있습니다.

7) 반복문 예고

- 반복문 : 같은 작업을 반복적으로 실행해주는 Javascript의 새로운 문법

```
var links = document.querySelectorAll('a');
var i = 0;
while (i<links.length) {
  links[i].style.color = 'powerblue';
  i=i+1;
}
```

먼저 첫 번째 줄에서는 이 페이지의 모든 a 태그를 가져옵니다. 그리고 그 다음 줄부터는 반복문을 이용해서 각각의 a 태그들의 color을 powerblue로 바꿔주는 것입니다.

8) 배열

- 배열(Array) : 프로토타입으로 탐색과 변형 작업을 수행하는 메소드를 갖는, 리스트와 비슷한 객체, 배열의 길이와 요소의 자료형은 고정되어 있지 않습니다.

```
var fruits = ["apple", "banana"];
```

배열은 다음과 같이 대괄호로 표시합니다. 그리고 이 대괄호 안에 넣을 값들을 콤마(,)로 표시해서 넣어줍니다. 그리고 이를 변수 fruits에 넣어줍니다. 즉 과일을 fruits라는 상자에 넣어준 것입니다.

- 배열의 값에 접근하기

배열에 들어있는 내용을 꺼내기 위해서는

```
document.write(fruits[0]);
```

화면에 apple이 출력되게 됩니다. 0 대신 1을 써주면 banana가 출력됩니다. 즉, 앞에서부터 0번째, 1번째, ... 로 순서를 매겨서 그 번호에 해당하는 값을 출력하는 것입니다.

- 배열의 길이

```
document.write(fruits.length);
```

배열 뒤에 .length를 쓰면 그 배열의 길이를 나타내줍니다. 이 때 출력값은 2가 되죠. 즉 배열에서 내용을 꺼낼 때에는 0부터 순서를 매기지만, 길이를 출력할 때에는 1부터 세서, 2개의 값이 있으니 2를 출력해주는 것입니다.

- 배열에 값 추가하기

```
fruits.push("coconut");
```

이렇게 하면 fruits라는 배열의 맨 뒤에 "coconut"이라는 값을 추가해줍니다. 즉, fruits는 ["apple", "banana", "coconut"]이 되는 것입니다.

9) 반복문

- 반복문이 필요한 이유

```
document.write('<li>1</li>');
document.write('<li>2</li>');
document.write('<li>3</li>');
document.write('<li>4</li>');
```

이 코드를 실행하면 화면에 1, 2, 3, 4가 순서대로 뜰 겁니다. 각 줄의 코드가 순서대로 실행되는 것입니다. 그런데 이때 2번과 3번이 반복적으로 여러 번 실행되어야 한다고 생각해봅시다. 그러면 2번과 3번 코드를 여러 번 복사, 붙여넣기를 하면 되겠죠. 하지만 반복 횟수가 아주 커진다면 붙여넣기도 힘들고 코드를 수정하기도 힘들 것입니다. 이때 사용하는 것이 바로 반복문입니다.

- 반복문

while 반복문

```
while (???) {
  document.write('<li>2</li>');
  document.write('<li>3</li>');
}
```

이때 ???에는 if 문에서 살펴본 것과 같이 boolean 데이터 타입이 들어갑니다. 그리고 이 조건이 true라면 중괄호 안의 코드를 실행합니다. 이 코드를 다 실행하고 나면 다시 처음으로 돌아가서 ???에 들어간 boolean을 살펴봅니다. 이 boolean이 false가 될 때까지 반복해서 코드를 실행하는 것입니다. 즉, 반복문과 조건문은 순서대로 실행되는 프로그램의 실행 흐름을 제어하는 제어문입니다.

```
document.write('<li>1</li>');
var i = 0;
while (i < 3) {
  document.write('<li>2</li>');
  document.write('<li>3</li>');
  i = i + 1;
}
document.write('<li>4</li>');
```

초기 상태에서 i는 0입니다. 따라서 while 뒤의 괄호 안에 들어있는 i<3은 true가 됩니다. 그러면 코드가 한 번 실행되겠죠. 코드가 실행되면 i = i + 1에 의해서 i의 값은 1이 됩니다. 이 과정을 3번 반복하면 i는 3이 됩니다. 그렇게 되면 i<3이 false가 되고, 반복문이 끝나고 마지막 8번째 줄이 실행되게 됩니다.

10) 배열과 반복문

- 배열과 반복문

```
fruits = ["apple", "banana", "coconut"];
```

여기에 있는 값들을 하나씩 꺼내서 HTML 리스트로 나타내보도록 하겠습니다. 반복문을 이용해서 말입니다. 여기에 있는 모든 원소들을 write하기 위해서는 3번의 반복이 필요할 겁니다. 그러므로 3번 반복할 수 있는 while문을 사용합니다.

```
var i = 0;
while (i < 3) {
  document.write('<li></li>');
  i = i + 1;
}
```

이제 li 태그 사이에 각각의 원소를 넣어봅시다. 이때 이 i라는 변수가 0부터 2까지 바뀌기 때문에, i를

잘 활용하면 배열에서 원소를 순서대로 하나씩 꺼낼 수 있을 겁니다.

```
var i = 0;
while (i < 3) {
  document.write('<li>'+fruits[i]+'</li>');
  i = i + 1;
}
```

하지만 여기에서 우리가 fruits라는 배열 안에 "durian"이라는 원소를 하나 더 추가했다고 해 봅시다. 그러면 이 반복문은 4번 실행되어야 끝까지 출력할 수 있겠죠. 하지만 지금은 3으로 설정되어있기 때문에 durian이 출력되지 않을 겁니다. 3 대신 fruits라는 배열의 길이를 입력해주면 fruits라는 배열에 원소가 추가되거나 삭제되어도 자동으로 반복 횟수가 조정될 겁니다.

```
var i = 0;
while (i < fruits.length) {
  document.write('<li>'+fruits[i]+'</li>');
  i = i + 1;
}
```

정리하자면, 배열을 사용하면 순서대로 연관된 데이터를 저장할 수 있고, 반복문을 사용하면 순서대로 이 배열의 원소를 하나씩 꺼내서 처리할 수 있기 때문에 이 둘을 함께 사용하면 아주 좋다는 것을 알 수 있습니다.

11) 배열과 반복문의 활용

- querySelectorAll이 찾은 모든 a 태그를 배열 형태로 alist에 저장하게 됩니다. 그렇다면 이제 이렇게 만든 배열과 반복문을 사용해서 각각의 태그의 스타일을 지정해주도록 합시다. while문을 사용해서 스타일을 바꿔주려고 하는데, alist의 길이만큼 실행되면 될 겁니다. 그렇다면 다음과 같이 써 주면 되겠습니다.

```
var i = 0;
while (i < alist.length) {

}
```

이 while문 안에 각각의 스타일을 지정하는 코드를 넣어주도록 합시다. 먼저 alist의 i번째 원소를 가져오면 되겠죠. 그리고 이렇게 가져온 태그의 스타일을 지정해주면 됩니다. 마지막에 i에 1을 더해주는 것도 잊으면 안됩니다.

```
var i = 0;
while (i < alist.length) {
  alist[i].style.color = 'powderblue';
  i = i + 1;
}
```

이 코드를 실행해보면 문서 안의 모든 a 태그의 색깔이 powderblue로 바뀌는 것을 볼 수 있을 겁니다.

3. Javascript 함수

1) 함수 예고

- 함수 : 하나의 <input> 태그 안에 아주 긴 Javascript 코드가 들어있습니다. 만약 이러한 토글을 여러 개 복사해서 만든다면 코드가 아주 길어질 것입니다. 그 뿐만 아니라 만약 약간의 수정을 하고 싶다고 한다면 복사된 모든 <input> 태그를 찾아서 하나하나 수정해야합니다. 이때 사용하는 것이 바로 함수이다.

```
<script>
```

```
  function nightdayhandler(self) {
    // 아주 긴 Javascript 코드를 여기 넣어줍니다.
  }
</script>
```

영어로는 function이라고 부르는데, 이 부분을 함수로 쓰겠다는 의미를 담아서 앞에 function이라는 키워드를 붙여줍니다. 그럼 이제 이렇게 새로운 이름을 붙여 주었으니, <input> 태그의 onclick 안에는 긴 코드를 다시 쓸 필요 없이, 붙여준 이름을 써 주기만 하면 됩니다. 다음과 같은 코드를 사용하면 됩니다.

```
<input id='night_day' type='button' value='night' onclick='nightdayhandler(this);'>
```

- 함수의 장점 : 코드의 유지보수가 쉬워집니다. 하나의 함수를 여러군데에서 사용할 때, 이를 하나하나 바꿔줄 필요 없이 함수를 만들어 준 곳에서만 바꿔주면 된다는 것입니다. 또한, 코드의 길이가 짧아집니다. 같은 코드가 계속해서 반복되는 것을, 딱 한 번만 써줌으로 인해서 웹페이지의 크기를 줄여줄 수 있고, 전송할 때 훨씬 유리해집니다. 마지막으로 같은 함수를 사용하면 두 코드가 논리적으로 같다는 것을 한 번에 알 수 있고, 적절한 이름을 붙여주면 이 코드가 어떤 일을 하는지 한 눈에 알 수 있다는 장점도 있습니다.

2) 함수

- 함수의 기본 문법

```
document.write('1');
document.write('2');
document.write('3');
document.write('4');
document.write('2');
document.write('3');
```

2를 쓰는 코드와 3을 쓰는 코드가 두 번 반복됨에도 불구하고, 그 둘 사이에 4를 쓰는 코드가 존재하기 때문에 반복문을 쓰기가 어려워집니다. 이럴 때 사용하는 것이 바로 함수입니다.

```
function two() {
  document.write('2');
  document.write('3');
}
```

이 코드에 two라는 이름을 붙여주었으니, 이 이름을 사용하면 되겠죠. 다음 코드와 같이 쓰게 되면 원래와 똑같은 기능을 하면서도 간결하게 나타낼 수 있습니다.

```
document.write('1');
two();
document.write('4');
two();
```

이 코드를 실행하게 되면, two()를 만날 때마다 위에서 정의된 two라는 함수의 코드를 실행하게 됩니다.

3) 함수-매개변수와 인자

- 함수의 입력과 출력 : 함수는 자판기에 비유할 수 있습니다. 마치 버튼이 1개만 있는 자판기와 같은 것이죠. 항상 같은 코드를 반환해주는 것입니다. 자판기의 버튼을 여러 개로 늘려보겠습니다. 원하는 코드에 대해 알려주면, 이에 해당하는 적절한 코드를 반환해주는 함수이죠. 이를 입력과 출력이라고 부릅니다.
- 매개변수와 인자 : 즉, 함수는 입력과 출력으로 이루어져 있는 것입니다. 이 때 입력에 해당하는 것을 매개변수Parameter)와 인자(Argument)라고 부르고, 출력에 해당하는 것을 리턴(Return)이라고 부릅니다.

```
function onePlusOne() {
```

```
  document.write(1+1);
}
```
이 함수는 항상 1과 1을 더해서 출력해주는 함수입니다. 하지만, 입력값을 받아서 거기에 따라서 다른 결과를 출력하도록 만들어 봅시다. 예를 들어서 sum(2,3)을 한다면 5가 출력되도록, sum(1,6)를 하면 7이 출력되려면

```
function sum(left, right) {
  document.write(left + right);
}
```
이때 sum의 괄호 안에 들어오는 두 숫자를 각각 left, right라는 변수에 넣는 것이죠. 이때 이러한 변수를 매개변수라고 부릅니다. 이제 이렇게 정의한 함수를 사용해봅시다. 다음과 같이 사용할 수 있습니다.

```
sum(2,3);
```
이때 sum의 괄호 안에 넣어서 함수로 전달해주는 저 숫자 2, 3을 보고 인자라고 부릅니다.

4) 함수-리턴

- 리턴 : 콘솔에 1+1을 치고 엔터를 누르면 2가 나온다. 이때 1+1은 2의 표현식이라고 부릅니다.

```
function sum(left, right) {
  document.write(left + right);
}
```
sum 함수를 실행시키면 document.write를 사용해서 바로 화면에 출력 해주었습니다. sumColorRed라는 새로운 함수를 만들어서 빨간색으로 write해주는 작업을 진행해 주어야합니다. 이렇게 하면 필요할 때마다 함수를 계속해서 만들어 주어야 하기 때문에 굉장히 불편할 겁니다. 그렇기 때문에 이 함수를 약간 바꾸어서 계산한 값의 표현식으로 만들어봅시다. 콘솔에 1+1을 치면 2가 나오는 것처럼 만들어주는 것입니다.

```
document.write(sum(2,3)+'<br>');
document.write('<div style="color:red">'+sum(2,3)+'</div>');
```
즉, sum 자체가 어떤 write를 수행하는 것이 아닌, 적절한 값을 돌려주는 것입니다.

```
function sum(left, right) {
  return left + right;
}
```

5) 함수 활용

- 함수의 활용 : 함수는 코드 리팩토링을 할 때 자주 사용하는 방법 중 하나입니다.

```
<script>
  function nightdayhandler() {
    if(this.value === 'night') {
    // 생략
    // 아주 긴 Javascript 코드를 여기 넣어줍니다.
  }
</script>
<input id='night_day' type='button' value='night' onclick='nightdayhandler();'>
```
하지만 이 상태로 코드를 실행해보면 정상적으로 작동하지 않는 것을 볼 수 있습니다. 코드 내에 포함되어 있는 this 때문입니다. this는 그 코드가 포함되어 있는 태그를 가리키는데, 지금은 this가 정상적으로 버튼을 가리키지 못하고 있기 때문입니다.

```
<script>
```

```
  function nightdayhandler(self) {
    if(self.value === 'night') {
    // 생략
    // 아주 긴 Javascript 코드를 여기 넣어줍니다.
  }
</script>
<input id='night_day' type='button' value='night' onclick='nightdayhandler(this);'>
```
즉, 함수에서는 this를 self라는 매개변수에 받아와서, 이를 이용해서 나머지 코드를 진행하는 것입니다.

4. Javascript 객체 소개

1) 객체 예고

- 객체 : 객체는 함수의 기반 위에서 존재하는 개념입니다. 서로 연관된 함수와 변수가 아주 많아지면 이를 정리하기 위해서 사용하는 것입니다.

```
function LinksSetColor(color){
  var alist = document.querySelectorAll('a');
  var i = 0;
  while(i < alist.length) {
    alist[i].style.color = color;
    i = i + 1;
  }
}
```

이제 하이퍼링크의 색깔을 바꾸려면

```
setColor("powderblue");
```

같은 방법으로 배경과 글자의 색을 바꾸는 코드도 따로 함수로 만들어 봅시다. 이때 함수 이름이 겹치지 않도록 하는데 주의해야 합니다.

```
function BodySetColor(color){
  document.querySelector('body').style.color = color;
}
function BodySetBackgroundColor(color){
  document.querySelector('body').style.backgroundColor = color;
}
```

이렇게 다양한 함수들이 존재하는 상황에서 우리는 객체를 사용하면 됩니다. 예를 들어 함수의 종류가 아주 많아지게 되면 이 함수들끼리 이름이 중복되지 않도록 만들게 하기 위해서 굉장히 복잡한 이름을 사용해야 합니다. 이때 객체를 사용하면 이 함수들을 비슷한 것들끼리 그룹으로 만들어 묶어줄 수 있습니다. 이렇게 나뉜 함수들은 서로 다른 그룹끼리는 이름이 겹쳐도 괜찮습니다.

비슷한 예시로 document.querySelector('body');

여기에서 document가 바로 객체이고, querySelector가 document라는 객체에 속해 있는 함수라고 생각할 수 있는 것입니다. 이렇게 객체에 속해 있는 함수들은 메소드(Method)라는 별도의 이름으로 부르게 됩니다.

2) 객체(쓰기와 읽기)

- 객체의 쓰기와 읽기 : 객체에는 순서가 없는 대신 각각에 이름이 붙어 있습니다. 이름을 이용해서 값들을 꺼내고 넣는 것입니다.

```
var coworkers= {
  "programmer": "egoing",
  "designer": "leezche"
};
```
객체를 만들 때에는 배열과는 달리 중괄호를 사용합니다. 그리고 각 요소들은 각각 이름과 값으로 이루어져 있죠. 예를 들면 egoing이라는 값에는 programmer이라는 이름표가 붙어있는 것입니다.
```
document.write(coworkers.programmer)
document.write(coworkers["programmer"])
```
이렇게 쓰면 coworkers라는 객체 안에서 programmer이라는 이름을 가진 값을 가져와서 출력하게 됩니다. 첫 번째 줄과 두 번째 줄의 코드 모두 같은 결과를 출력하게 됩니다.
```
coworkers.bookkeeper = "duru";
coworkers["bookkeeper"] = "duru";
```
이런 코드를 실행하게 되면 coworkers라는 객체 안에 bookkeeper이라는 이름으로 duru라는 값을 넣는 것입니다. 두 줄 모두 같은 작업을 실행하게 됩니다. 하지만 만약에 이름 안에 공백이 있다면 첫 번째 줄의 코드처럼 쓰면 오류가 나기 때문에 두 번째 줄처럼 써 주어야 합니다.

3) 객체(순회)

- 객체의 순회 : 객체에 있는 모든 값들을 가져오는 방법, 배열에서는 반복문을 사용했었고, 객체에서는 for in이라는 것을 사용하게 됩니다.

```
var coworkers = {
  "programmer": "egoing",
  "designer": "leezche"
};

for(var key in coworkers) {
  document.write(key+'<br>');
}
```
for을 쓰면 coworkers에 있는 이름들을 하나씩 가져오게 됩니다. 그리고 이 이름을 차례대로 key에 넣어줍니다. 그러면 그 key들이 차례대로 출력되겠죠. 결과적으로 programmer와 designer가 출력될 겁니다.
```
for(var key in coworkers) {
  document.write(coworkers[key]+'<br>');
}
```

4) 객체(프로퍼티와 메소드)

- 객체의 메소드 : 객체에는 다양한 것들을 담을 수 있고, 함수도 담을 수 있습니다.

```
coworkers.showAll = function() {
  for (var key in coworkers) {
    document.write(key + ' : ' + coworkers[key] + '<br>');
  }
}
```
즉, coworkers에서만 사용할 수 있는 showAll이라는 메소드를 추가하게 된 것입니다. 하지만 이 방법은 그렇게 좋은 방법은 아닙니다. 왜냐하면 cowerkers라는 변수 이름이 바뀌면 함수를 수정해야 하기 때문이죠. 이 때 사용하는 것이 바로 this입니다. coworkers 대신에 이 메소드가 쓰인 객체를 가리키는 this

를 사용해주면 됩니다.

```
coworkers.showAll = function() {
  for (var key in this) {
    document.write(key + ' : ' + this[key] + '<br>');
  }
}
```

- 객체의 프로퍼티 : 객체에 해당하는 함수들은 메소드라 하고 객체에 해당하는 변수

5) 객체의 활용

- 객체의 활용

```
var Body = {
  setColor: function (color) {
    document.querySelector('body').style.color = color;
  },
  setBackgroundColor: function (color) {
    document.querySelector('body').style.backgroundColor = color;
  }
}
var Links = {
  setColor: function (color) {
    var alist = document.querySelectorAll('a');
    var i = 0;
    while (i < alist.length) {
      alist[i].style.color = color;
      i = i + 1;
    }
  }
}
```

Body와 Links에 모두 setColor이라는 함수가 존재하게 됩니다. 하지만 이 둘을 사용할 때에는 다음과 같이 다르게 사용하게 됩니다.

```
Body.setColor('black');
Links.setColor('powderblue');
```

함수의 이름이 같아도 다른 객체에 소속된 메소드이기 때문에 충돌이 일어나지 않는 것입니다.

5. Javascript 객체 기본

1) 실습 준비

- Node js 에서 Java Script를 사용할 때 : js 파일을 만든 후, 콘솔 창에 node 파일명.js 을 입력하면 결과를 확인할 수 있습니다.
- 웹 브라우저에서 Javascript를 사용할 때 : html 파일에서 자바스크립트를 사용할 때

```
<html>
  <body>
    <script>
      console.log('Hello');
```

```
    </script>
  </body>
</html>
```

혹은 html 파일에서 Javascript 파일을 호출해 사용할 수 있습니다.

```
<html>
  <body>
    <script src="hello.js"></script>
  </body>
</html>
```

웹 브라우저를 연 후, 실행시킨 페이지에서 개발자 도구를 실행시킵니다. console 탭으로 이동하고, (맥 - cmd + o / 윈도우 - ctrl + o)를 눌러 html 파일을 실행 시킵니다. 그러면 콘솔 창에 Hello라는 문구가 뜬다.

https://nodejs.org/ko/download/

2) 객체의 기본

- 객체 생성 : 배열을 만듭니다.

```
var memberArray = ['egoing', 'graphittie', 'leezhce'];
//배열에서는 값에 접근할 때 []를 사용합니다.
console.log("memberArray[2]", memberArray[2]);
```

목록만 있으면 되는 경우에는 배열을 쓰지만, 각각의 데이터가 어떤 데이터인지를 풍부하게 설명해야 하는 경우는 객체를 사용하게 됩니다. 객체를 생성합니다.

```
var memberObject = {
  // 원소의 이름 : 원소 값
  manager: 'egoing',
  developer: 'graphittie',
  designer: 'leezhce'
}
//객체에서는 값에 접근할 때 .를 사용합니다.
memberObject.designer = 'leezche';
console.log('memberObject.designer', memberObject.designer);
//[]를 이용해 접근할 수도 있습니다.
console.log("memberObject.['designer']", memberObject['designer']);
```

- 객체 수정 : designer의 이름을 수정합니다.

```
var memberObject = {
  // 원소의 이름 : 원소 값
  manager: 'egoing',
  developer: 'graphittie',
  designer: 'leezhce'
}
memberObject.designer = 'leezhe';
```

- 객체 삭제 : delete를 사용해 객체에서 manager를 삭제해봅니다.

```
delete memberObject.manager
console.log('after delete memberObject.manager', memberObject.manager);
```

결과 값이 undefined가 나오는 것을 볼 수 있습니다.

3) 객체와 반복문

- 배열에서의 반복문 : 가장 기본적인 While 문을 사용해보면, I의 값은 0부처 memberArray의 길이보다 1작은 값까지 증가하기 때문에 memberArray에 있는 값을 하나 하나 꺼내올 수 있게 됩니다. console.group를 사용하면 결과 값을 더 보기 좋게 정리할 수 있습니다.

```
var memberArray = ['egoing','graphittie','leezche'];
console.group('array loop');
var i = 0;
while(i<memberArray.length){
    console.log(i, memberArray[i]);
    i = i + 1;
}
console.groupEnd('array loop');
```

- 객체에서의 반복문 : 배열에서 사용하는 for문과는 문법이 조금 다른 for-in문을 사용하면

```
console.group('object loop');
var memberObject = {
    manager: 'egoing',
    developer:'grphittie',
    designer: 'leezche'
}

for(var name in memberObject ){ // (현재 원소의 이름이 들어갈 변수) in (객체)
    //객체에 있는 원소의 개수만큼 중괄호가 실행됩니다.
    console.log(name);
}
console.groupEnd('object loop');
```

객체의 각 속성(객체가 가지고 있는 원소들)을 출력하면

```
console.group('object loop');
var memberObject = {
    manager: 'egoing',
    developer:'grphittie',
    designer: 'leezche'
}

for(var name in memberObject ){ // (현재 원소의 이름이 들어갈 변수) in (객체)
    //객체에 있는 원소의 개수만큼 중괄호가 실행됩니다.
    console.log(name, memberObject[name]);
}
console.groupEnd('object loop');
```

4) 객체의 쓰임

- 날짜와 관련된 기능, 수학과 관련된 기능 등 여러 가지 기능들이 존재합니다. 이러한 기능들을 잘 정돈하기 위해서 자바스크립트 개발자들은 이러한 객체를 이용합니다. 예를 들어 Math라는 객체에는 수학과 관련된 여러 함수들이 그룹화되어 있습니다.

```
console.log("Math.PI", Math.PI); // 파이 값을 출력합니다.
console.log("Math.random()", Math.random()); // 랜덤 값을 출력합니다.
console.log("Math.floor(3.9)", Math.floor(3.9)); // 값을 반올림합니다.
```
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects

5) 객체 만들어 보기

- 메소드 : 객체 안에 포함된 함수

```
var MyMath = {
    PI: Math.PI,
    random:function(){
        return Math.random();
    },
    floor:function(val){
        return Math.floor(val);
    }
}

console.log("MyMath.PI", MyMath.PI);
console.log("MyMath.random()", MyMath.random());
console.log("MyMath.floor(3.9)",MyMath.floor(3.9));
```

객체를 사용하지 않는다면

```
var MyMath_PI = Math.PI;
function MyMath_random(){
    return Math.random();
}
function MyMath_floor(val){
    return Math.floor(val);
}
```

객체를 사용하면 관련된 기능을 그룹화하여 편리하게 사용할 수 있습니다.

6) this

- this : 객체란 서로 연관된 변수와 함수를 그룹화해 이름을 붙인 것으로, 한국어와 영어에 자기 자신을 가리키는 대명사가 있듯이 프로그래밍에서도 자기 자신을 가리키는 표현이다.

객체를 만들고 sum 메소드를 호출해봅니다.

```
var kim = {
    name: 'kim',
    first: 10, //첫번째 게임 점수
    second: 20,  // 두번째 게임 점수
    sum:function(f,s){ // 게임 접수 합계 함수
        return f+s;
    }
}

console.log("kim.sum(kim.first,kim.second)",kim.sum(kim.first,kim.second));
```

이미 객체 내부에서 first와 second를 알고 있기 때문에 굳이 한 번 더 언급할 필요가 없습니다. sum 인자를 생략할 수 있도록 this를 이용해 객체를 수정할 수 있습니다. 어떤 메소드에서 그 메소드가 속해 있는 객체를 가리키는 특수한 키워드를 this라고 합니다.

```
var kim = {
    name: 'kim',
    first: 10, //첫번째 게임 점수
    second: 20,  // 두번째 게임 점수
    sum:function(){ // 게임 접수 합계 함수
        return this.first+this.second;
    }
}

console.log("kim.sum()",kim.sum());
```

this라는 키워드의 의미는 상당히 중요합니다.

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/this

7) constructor의 필요성

```
var kim = {
    name: 'kim',
    first: 10, //첫번째 게임 점수
    second: 20,  // 두번째 게임 점수
    sum:function(){ // 게임 접수 합계 함수
        return this.first+this.second;
    }
}

var lee = {
    name: 'lee',
    first: 10,
    second: 10,
    sum:function(){
        return this.first+this.second;
    }
}
console.log("kim.sum()",kim.sum());
console.log("lee.sum()",lee.sum());
```

이전 코드에 객체를 추가한 것으로, 프로그래밍 상으로는 문제 없지만 객체 내부의 내용이 바뀌면 같은 동작을 하는 모든 객체의 내용을 바꿔야 한다는 단점이 있습니다.

8) constructor의 사례 - date

- 자바스크립트 내장 객체에는 시간과 관련된 date 객체가 있습니다. new 키워드를 사용해 새로운 date 객체를 생성하면

```
var d1 = new Date('2019-4-10'); //2019년 4월 10일의 값을 가지는 Date 객체를 생성합니다.
console.log('d1.getFullYear()',d1.getFullYear()); // 해당 객체의 년도를 출력합니다.
```

```
console.log('d1.getMonth()',d1.getMonth()); //0부터 카운트하여 해당 객체의 월을 출력합니다.
```

이처럼 객체를 만드는 공장이 있다면 원하는 값을 가지는 객체를 양산해낼 수 있게 됩니다.

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Date

9) constructor 만들기

- 생성자 : 객체 생성 함수를 만들면

```
function Person(){
    this.name = 'kim';
    this.first = 10;
    this.second = 20;
    this.third = 30;
    this.sum = function(){
        return this.first+this.second+this.third;
    }
}

console.log('Person()', Person());  // undefined
console.log('new Person()', new Person()); // 객체 생성
```

그냥 함수를 호출할 경우 일반 함수 취급되지만, new라는 키워드를 붙일 경우 객체를 생성하는 생성자가 됩니다. 그 생성자를 constructor라고 합니다. 생성자를 이용해 새로운 객체를 생성하면

```
var kim = new Person();
var lee = new Person();
console.log("kim.sum()",kim.sum());
console.log("lee.sum()",lee.sum());
```

각각 객체가 다른 값을 갖게하려면 이전에 사용했던 date 함수처럼 생성자 함수가 실행될 때 입력 값을 주도록 할 수 있습니다.

```
function Person(name,first,second,third){
    this.name = name;
    this.first = first;
    this.second = second;
    this.third = third;
    this.sum = function(){
        return this.first+this.second+this.third;
    }
}


var kim = new Person('kim',10,20,30);
var lee = new Person('lee',10,10,10);
```

이것이 객체를 찍어내는 방법인 constructor 함수를 만드는 방법입니다.

10) prototype이 필요한 이유

- Prototype : 프로토타입은 원형이라는 뜻으로, 자바스크립트에서는 중급에서 고급으로 넘어가는 길목에 있다고 할 수 있을 정도로 중요한 개념이며 자바스크립트를 prototype based language라고 합니다.

```
function Person(name,first,second,third){
    this.name = name;
    this.first = first;
    this.second = second;
    this.third = third;
    this.sum = function(){
        return this.first+this.second+this.third;
    }
}


var kim = new Person('kim',10,20,30);
var lee = new Person('lee',10,10,10);
```

생성자 함수에서는 새로운 객체가 생성될 때마다 sum이라는 내부 메소드가 새롭게 생성되고 있습니다. 그만큼 메모리 낭비가 발생해 성능이 떨어지게 됩니다. 또, sum이라는 메소드의 내용을 수정하고 싶은 경우 만들어진 객체만큼 수정 작업을 반복해야 한다는 문제가 있습니다. 즉, 생산성이 떨어지게 됩니다.

https://developer.mozilla.org/ko/docs/Learn/JavaScript/Objects/Object_prototypes

11) prototype을 이용해서 재사용성을 높이기

- Person 생성자에 prototype에 sum이라는 함수를 정의하면

```
function Person(name,first,second,third){
    this.name = name;
    this.first = first;
    this.second = second;
    this.third = third;

}
Person.prototype.sum = function(){
    return this.first+this.second+this.third;
}

var kim = new Person('kim',10,20,30);
var lee = new Person('lee',10,10,10);
console.log("kim.sum()",kim.sum());
console.log("lee.sum()",lee.sum());
```

생성자 함수 안에 메소드를 정의하는 코드가 들어있지 않기 때문에 객체가 생성될 때마다 호출되지 않고 한 번만 생성하게 됩니다. 즉, 메모리를 절약할 수 있습니다. 만약 sum의 내용이 수정된다해도 한 번만 수정하면 됩니다.

```
Person.prototype.sum = function(){
    return 'prototype : ' + (this.first+this.second+this.third);
}

var kim = new Person('kim',10,20,30);
var lee = new Person('lee',10,10,10);
```

```
console.log("kim.sum()",kim.sum());
console.log("lee.sum()",lee.sum());
```

여러 개의 객체가 하나의 함수를 공유하므로써 성능을 높이고 메모리를 절약할 수 있습니다. 만약 하나의 객체에서만 sum이라는 함수를 다르게 동작시키고 싶다면 kim이라는 객체에 sum 메소드를 추가합니다.

```
function Person(name,first,second,third){
    this.name = name;
    this.first = first;
    this.second = second;
    this.third = third;

}
Person.prototype.sum = function(){
    return 'prototype : ' + (this.first+this.second+this.third);
}

var kim = new Person('kim',10,20,30);

kim.sum = function(){
    return 'this : ' + (this.first+this.second+this.third);
}
var lee = new Person('lee',10,10,10);
console.log("kim.sum()",kim.sum());
console.log("lee.sum()",lee.sum());
```

kim과 lee에서 각각 sum을 호출한 결과가 다르게 나오는 것을 확인할 수 있습니다. 자바스크립트는 객체에서 어떠한 메소드 또는 속성을 출력할 때 해당 객체가 그 메소드 또는 속성을 가지고 있는지를 확인합니다. 만약 가지고 있다면 객체 내의 메소드 또는 속성을 호출하고 없다면 이 객체의 생성자의 prototype에 해당 메소드 또는 속성이 정의 되어 있는지를 확인합니다.

- 생성자를 이용한 객체 생성 : 객체의 속성들(변수들)은 생성자 함수 안에 넣는 것이 일반적입니다. 객체의 메소드들은 생성자의 prototype에 추가하는 것이 일반적인 패턴입니다.

```
function Person(name,first,second,third){
    this.name = name;
    this.first = first;
    this.second = second;
    this.third = third;

}
Person.prototype.sum = function(){
    return 'prototype : ' + (this.first+this.second+this.third);
}
```

12) Classes

- classes : 자바스크립트는 가장 바르게 발전하는 언어 중에 하나입니다. 전통적인 객체 지향의 문법을 채택하므로써 이미 객체 지향을 사용할 수 있는 사람들이 문법적인 거부감 없이 자바스크립트에 안착할 수 있도록 합니다. 그 중에 하나가 class입니다. 다른 많은 언어들은 객체를 만드는 공장으로써 class를

지원하고 있습니다. constructor의 대체재라고 할 수 있다.
- 호환성과 치환 : 자바스크립트는 ECMA script라는 표준을 따르는데 class는 ECMA script 6부터 도입된 문법입니다. 이전 버전에서는 동작하지 않는다는 단점을 가지고 있습니다. 하지만 오늘날 많은 웹브라우저와 Node.js 와 같은 플랫폼들이 ECMA script 6 이상의 버전을 지원하고 있기 때문에 크게 문제가 되지는 않습니다. 이러한 의사 결정은 통계를 기반으로 해야합니다. https://caniuse.com/ 는 이러한 통계를 제공하는 사이트입니다. ES6 classes를 검색하면 해당 문법을 지원하는 웹 브라우저들의 정보를 보여줍니다. 자바스크립트는 원래 객체 지향 언어였고 prototype 기반 언어입니다. 그리고 새롭게 도입된 문법은 이미 존재하던 기능을 변형한 문법입니다. 즉, 기존에 존재하는 문법으로도 똑같은 기능을 낼 수 있습니다. https://babeljs.io/ 은 새로운 문법을 기존의 문법으로 치환해주는 javascript compiler입니다. 웹 사이트를 이용하거나 babel에서 제공하는 자동화된 command line을 사용해서 기존의 문법으로 치환할 수 있습니다.

13) Classes의 생성

- class를 이용한 객체 생성 : 기존에 만들었던 Person 생성자 함수와 동일하게 동작하는 class를 정의해봅시다. 생성자 함수의 역할은 두 가지 있습니다. 객체를 만든다, 객체의 초기 상태를 정의한다. 이것을 class에서는 어떻게 하는지, 우선 객체를 생성하면

```
class Person{

}
var kim = new Person();
console.log('kim',kim);
```

코드를 실행시켜보면 객체가 생성된 것을 확인할 수 있습니다.

14) class의 constructor function

- constructor 함수 : class는 객체의 초기값을 지정하기 위해서 객체가 생성될 때 실행되기로 약속된 함수입니다. 이 함수를 이용해 객체의 초기값을 설정할 수 있습니다. 자바스크립트는 객체를 생성할 때 자동으로 constructor 함수를 호출합니다.

```
class Person{
    constructor(){ // 약속된 이름으로 바꾸면 안됩니다.
        console.log('constructor');
    }
}
var kim = new Person();
console.log('kim',kim);
```

constructor 함수에 입력을 받아 객체의 초기값을 설정하면

```
class Person{
    constructor(name,first,second){ // 약속된 이름으로 바꾸면 안됩니다.
        this.name = name;
        this.first = first;
        this.second = second;
        console.log('constructor');
    }
}
var kim = new Person('kim',10,20);
```

```
console.log('kim',kim);
```

15) 메소드 구현

- 메소드 만들기 : prototype을 이용해 추가합니다.

```
class Person{
    constructor(name,first,second){ // 약속된 이름으로 바꾸면 안됩니다.
        this.name = name;
        this.first = first;
        this.second = second;
        console.log('constructor');
    }
}
Person.prototype.sum = function(){
    return 'prototype : ' + (this.first+this.second+this.third);
}
var kim = new Person('kim',10,20);
console.log('kim',kim);
```

class 내부에 정의하기

```
class Person{
    constructor(name,first,second){ // 약속된 이름으로 바꾸면 안됩니다.
        this.name = name;
        this.first = first;
        this.second = second;
        console.log('constructor');
    }
    sum(){
        return 'prototype : ' + (this.first+this.second);
    }
}
var kim = new Person('kim',10,20);
console.log('kim',kim);
console.log("kim.sum()",kim.sum());


var lee = new Person('lee',10,10);
console.log('lee',lee);
console.log("lee.sum()",lee.sum());
```

같은 class에 속해 있는 모든 객체들이 공유하는 메소드임을 알 수 있습니다. 특정 객체의 메소드만 수정하고 싶다면 기존과 똑같은 방법을 사용하면 안됩니다.

```
var kim = new Person('kim',10,20);
kim.sum = function(){
    return 'this : ' + (this.first+this.second);
}
```

어떤 객체의 특성을 호출하면 자바스크립트는 그 객체가 해당 특성을 가지고 있는지 확인하고 있다면 그

특성을 호출합니다. 만약 없다면 그 객체가 속해 있는 class에서 해당 특성을 가져와 호출합니다.

6. Javascript 객체 고급

1) 상속

- Person class에 평균을 구하는 avg 메소드를 추가하면

```
class Person{
    constructor(name,first,second){
        this.name = name;
        this.first = first;
        this.second = second;
        console.log('constructor');
    }
    sum(){
        return this.first+this.second;
    }
    avg(){
        return (this.first+this.second)/2;
    }
}
```

class에 어떤 기능을 추가하고 싶은데 만약 남이 짠 코드라 수정할 수 없는 경우나 추가하고 싶은 기능이 거의 사용되지 않는 경우 전체 코드를 수정하는 것은 부담스러운 일이 될 수 있습니다. Person은 이전처럼 되돌려놓고 새로운 PersonPlus라는 class를 새로 정의하면

```
class Person{
    constructor(name,first,second){
        this.name = name;
        this.first = first;
        this.second = second;
        console.log('constructor');
    }
    sum(){
        return this.first+this.second;
    }
}
class PersonPlus{
    constructor(name,first,second){
        this.name = name;
        this.first = first;
        this.second = second;
        console.log('constructor');
    }
    sum(){
        return this.first+this.second;
    }
```

```
    avg(){
        return (this.first+this.second)/2;
    }
}
var kim = new PersonPlus('kim',10,20);
console.log("kim.sum()",kim.sum());
console.log("kim.sum()",kim.avg());
```

Person의 내용이 PersonPlus에 중복되고 있다는 아쉬움이 있습니다. 이 중복을 제거해주는 기능이 바로 상속입니다. PersonPlus가 Person을 상속하도록 수정하면

```
class PersonPlus extends Person{ //person이 personPlus에 상속됩니다.
    avg(){
        return (this.first+this.second)/2;
    }
}

var kim = new PersonPlus('kim',10,20);
console.log("kim.sum()",kim.sum());
console.log("kim.sum()",kim.avg());
```

위와 동일한 결과가 나오는 것을 확인할 수 있습니다. PersonPlus class는 avg 메소드를 제외한 모든 기능을 Person calss에서 가져오고 있기 때문에 Person class를 수정하면 PersonPlus를 사용하는 객체 모두가 변경되게 됩니다. 우리는 상속을 이용해 기존 클래스를 확장하여 중복을 제거하고 코드의 양을 줄였으며 공유 하는 코드를 수정하면 상속 받는 모든 객체들에 동시다발적으로 변화가 일어나도록해 유지보수의 편리성을 높였습니다. 이것이 상속입니다.

2) super

- PersonPlus class에만 third라는 새로운 인자를 추가하고 싶다면 PersonPlus에 Person의 기능을 모두 가져와 수정한다면 상속의 의의가 없어지게 되는 문제가 있습니다.

```
class PersonPlus extends Person{
    constructor(name,first,second,third){
        this.name = name;
        this.first = first;
        this.second = second;
        this.third = third;
    }
    sum(){
        return this.first+this.second+this.third;
    }
    avg(){
        return (this.first+this.second+this.third)/3;
    }
}
```

이럴 때 사용하는 키워드가 바로 super입니다. super을 이용하면 부모 클래스가 가지고 있는 기능을 실행할 수 있습니다. super 용법에는 두 가지가 있습니다.

```
// 1. 부모 클래스의 생성자 호출
```

```
super()

// 2. 부모 클래스
super.sum()
 만약  PersonPlus의 constructor가 실행되기 전에 부모 클래스의 기능이 먼저 실행되도록 한다면
class Person{
    constructor(name, first, second){
        this.name = name;
        this.first = first;
        this.second = second;
    }
    sum(){
        return this.first+this.second;
    }
}
class PersonPlus extends Person{
    constructor(name, first, second, third){
        super(name, first, second);
        this.third = third;
    }
    sum(){
        return super.sum()+this.third;
    }
    avg(){
        return (this.first+this.second+this.third)/3;
    }
}

var kim = new PersonPlus('kim', 10, 20, 30);
console.log("kim.sum()", kim.sum());
console.log("kim.avg()", kim.avg());
```

3) object_inheritance

- 객체 지향 프로그래밍 : 크게 두 가지 요소로 나눌 수 있다. 첫 번째는 객체를 만들어내는 공자, 설계도라고 할 수 있는 class, 두 번째는 class를 통해 만들어진 구체적인 객체가 있습니다. 이 두 가지가 어떻게 상호작용을 하느냐에 따라 상당히 다른 형태의 객체 지향 언어들이 만들어집니다. 주류인 자바의 객체 지향과 자바스크립트의 객체 지향은 다르다.
- 주류 객체 지향 언어에서의 상속 : 자바와 같은 언어에서 sub class가 super class의 기능을 물려 받기 위해서는 sub class가 super class의 자식이 되어야 합니다. 그리고 만들어진 sub class를 통해서 객체를 생성해냅니다. 따라서 이 객체가 어떠한 기능을 갖게 될 것인지는 class 단에서 결정됩니다. 객체가 다른 객체의 상속의 받는 등의 경우는 불가능 합니다.
- 자바스크립트에서의 상속 : 자바스크립트에도 class라는 키워드는 있지만 이것은 장식에 불과하고 내부 동작이 바뀐 것은 아닙니다. 자바스크립트는 이것보다 더 자유롭게 복잡하게 상속을 구현합니다. 여기 어떤 super object가 있고 이 객체의 기능을 상속을 받으면서 새로운 기능을 추가하고 싶은 sub object가

있다고 해봅니다. sub object는 super object로부터 바로 상속을 받을 수 있습니다. class가 상속을 받는 전통적인 방법과 달리 자바스크립트에서는 객체가 직접 다른 객체를 상속 받을 수 있고, 얼마든지 상속 관계를 바꿀 수 있습니다. 만약 super object로부터 상속을 받고 있는 sub object가 다른 객체로부터 상속을 받고 싶다면 링크만 바꿔주면 됩니다. 이러한 링크를 prototype link라고 합니다. 그리고 prototype link가 가리키는 객체를 prototype object라고도 합니다.

4) _proto_

- 간단한 객체를 만들어 상속해보면 _proto_라는 prototype link를 통해 객체를 상속 받을 수 있습니다.

```
var superObj = {superVal:'super'}
var subObj = {subVal:'sub'}
subObj.__proto__ = superObj;
console.log('subObj.subVal => ',subObj.subVal);
console.log('subObj.superVal => ',subObj.superVal);
```

객체의 속성을 바꿔도 __proto__의 속성은 바뀌지 않습니다. 그렇기 때문에 subObj.superVal 의 값을 바꿔도 superObj.superVal()의 값은 유지됩니다.

```
subObj.superVal = 'sub';
console.log('superObj.superVal => ', superObj.superVal);
//superObj.superVal =>  super
```

5) Object.create()

- Object.create : Object.create를 사용해서 객체를 상속하는 새로운 객체를 만들 수 있습니다. Object.create의 인자로 부모로 지정할 객체를 넣어 줍니다.

```
var superObj = {superVal:'super'}
var subObj = Object.create(superObj);
subObj.subVal = 'sub';

console.log('subObj.subVal => ',subObj.subVal);
console.log('subObj.superVal => ',subObj.superVal);

subObj.superVal = 'sub';
console.log('superObj.superVal => ', superObj.superVal);
```

debugger를 사용해 객체의 모습을 좀 더 자세히 살펴보겠습니다. 기존의 hello.html 파일을 수정합니다.

```
<html>
    <body>
        <script src="prototype-inheritance.js"></script>
    </body>
</html>
```

Object.create가 실행됐을 때 상태를 살펴보기 위해서 subObj 생성 부분 아래에 debugger라고 작성해 줍니다. debugger 키워드로 자바스크립트를 일시 중지시킬 수 있고 객체를 자세하게 들여다볼 수 있습니다.

```
var superObj = {superVal:'super'}
// var subObj = {subVal:'sub'}
// subObj.__proto__ = superObj;
```

```
var subObj = Object.create(superObj);
subObj.subVal = 'sub';
debugger;

console.log('subObj.subVal => ',subObj.subVal);
console.log('subObj.superVal => ',subObj.superVal);

subObj.superVal = 'sub';
console.log('superObj.superVal => ', superObj.superVal);
```

hello.html 파일을 실행시킨 후 개발자 도구를 켜고 새로고침을 하면 source 탭에 멈춰 있는 모습을 확인할 수 있습니다. Watch에서 subObj를 추가하면 쉽게 객체를 확인할 수 있습니다. subObj의 __proto__가 superObj로 설정 되어있는 것을 확인할 수 있습니다.

https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Object/create

6) 객체 상속의 사용

- __proto__ 를 사용해 객체를 상속 받는 객체를 만들고 메소드를 추가해봅시다.

```
kim  = {
    name: 'kim',
    first:10, second:20,
    sum:function(){return this.first+ this.second}
}

lee = {
    name:'lee',
    first:10,second:10,
    avg:function(){
        return (this.first+this.second)/2;
    }
}
lee.__proto__ = kim;
console.log('lee.sum()', lee.sum());
console.log('lee.avg()', lee.avg());
```

같은 내용을 object.create()를 사용해 만들어 봅시다.

```
kim  = {
    name: 'kim',
    first:10, second:20,
    sum:function(){return this.first+ this.second}
}

// lee = {
//     name:'lee',
//     first:10,second:10,
//     avg:function(){
//         return (this.first+this.second)/2;
```

```
//      }
// }
var lee = Object.create(kim);
lee.name = 'lee';
lee.first = 10;
lee.second = 10;
lee.avg = function(){
    return (this.first + this.second)/2;
}

lee.__proto__ = kim;
console.log('lee.sum()', lee.sum());
console.log('lee.avg()', lee.avg());
```

7) 객체와 함수

- 자바스크립트는에서 함수는 단독으로 쓰일 수도 있습니다. new가 앞에 있으면 객체를 만드는 생성자로 쓰일 수도 있고 call, bind 등 자유롭고 놀라운 사용법이 존재합니다.

8) call

- 연관 없는 두 객체와 공통으로 가지는 sum이라는 함수를 만들어 봅시다. 또 한 객체가 다른 객체를 상속하도록 __proto__를 지정해봅시다.

```
var kim = {name:'kim',first:10,second:20}
var lee = {name:'lee',first:10,second:10}
lee.__proto__ = kim

function sum(){
    return this.first + this.second;
}
```

sum이라는 함수는 어떤 객체에도 속해 있지 않은 채 생성되었지만 기괴하게도 객체들 안에 있는 first와 second라는 속성을 더하는 역할을 하고 있습니다. 자바스크립트의 모든 함수는 call이라는 메소드를 가집니다. 자바스크립트에서는 함수도 객체이기에 call 메소드의 인자로 객체를 지정하게 되면 해당 함수의 this는 인자로 받은 객체가 됩니다.

```
var kim = {name:'kim',first:10,second:20}
var lee = {name:'lee',first:10,second:10}
lee.__proto__ = kim

function sum(){
    return this.first + this.second;
}

//sum();
console.log("sum.call(kim)",sum.call(kim)); //sum.call(kim) 30
console.log("sum.call(lee)",sum.call(lee)); //sum.call(lee) 20
```

call은 여러 인자를 가질 수 있습니다. 첫 번째 인자는 this로 지정할 객체가 오고 그 뒤로는 함수의 인

자로 들어갈 값이 들어가게 됩니다.

```
var kim = {name:'kim',first:10,second:20}
var lee = {name:'lee',first:10,second:10}
lee.__proto__ = kim

function sum(prefix){
    return prefix+ (this.first + this.second);
}

//sum();
console.log("sum.call(kim)",sum.call(kim,'=> ')); //sum.call(kim) => 30
console.log("sum.call(lee)",sum.call(lee,': ')); //sum.call(lee) : 20
```

9) bind

- 만약 call처럼 실행할 때마다 this를 변경하는 것이 아니라 내부적으로 고정시키고 싶다면 bind를 사용합니다. bind는 호출한 함수를 변경하는 것이 아니라 인자로 받은 조건을 만족하는 새로운 함수를 리턴해줍니다.

```
var kim = {name:'kim',first:10,second:20}
var lee = {name:'lee',first:10,second:10}
lee.__proto__ = kim

function sum(prefix){
    return prefix+ (this.first + this.second);
}

//sum();
console.log("sum.call(kim)",sum.call(kim,'=> '));
console.log("sum.call(lee)",sum.call(lee,': '));

var kimSum = sum.bind(kim, '-> ');
console.log('kimSum()', kimSum());
```

10) prototype vs _proto_

- 자바스크립트의 함수는 객체입니다. 따라서 프로퍼티를 가질 수 있습니다.

```
function Person(){}
var Person = new Function();
```

Person이라는 함수를 생성하면 Person이라는 객체와 Person의 prototype 객체가 생성되게 됩니다.

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

그리고 두 객체는 서로 연관되어 있기 때문에 서로의 존재를 알아야 합니다. Person 객체의 prototype은 Person의 prototype 객체를 가리킵니다. Person의 prototype 객체도 Person에 속해있다는 것을 표

시하기 위해서 constructor 프로퍼티에 Person 객체를 기록합니다. 즉, 서로 참조하는 상태입니다. 여기에 sum이라는 메소드를 추가하면

Person.prototype.sum = function(){}

생성자를 이용해 새로운 객체를 생성해봅시다.

var kim = new Person('kim',10,20)

이 kim이라는 객체는 constructor 함수가 실행되면서 this의 값이 세팅된 결과 프로퍼티 값들이 생성이 되고 동시에 __proto__ 라는 프로퍼티도 생성이 됩니다. 자바스크립트는 __proto__의 값으로 해당 객체를 생성한 생성자의 prototype을 가리키게 합니다.

var lee = new Person('lee',10,10)

어떤 객체에서에서 메소드나 프로퍼티를 호출할 때 자바스크립트는 먼저 해당 객체에 호출하려는 값이 있는지 살피고 없다면 그 객체의 __proto__ 프로퍼티가 가리키는 prototype 객체에서 호출하려는 값을 찾습니다.

console.log(kim.name);

kim.sum();

https://developer.mozilla.org/ko/docs/Learn/JavaScript/Objects/Object_prototypes

https://developer.mozilla.org/ko/docs/Web/JavaScript/Inheritance_and_the_prototype_chain

11) 생성자를 통한 상속 - 소개

```
class Person{
    constructor(name, first, second){
        this.name = name;
        this.first = first;
        this.second = second;
    }
    sum(){
        return this.first+this.second;
    }
}
class PersonPlus extends Person{
    constructor(name, first, second, third){
        super(name, first, second);
        this.third = third;
    }
    sum(){
        return super.sum()+this.third;
    }
    avg(){
        return (this.first+this.second+this.third)/3;
    }
}

var kim = new PersonPlus('kim', 10, 20, 30);
console.log("kim.sum()", kim.sum());
console.log("kim.avg()", kim.avg());
```

12) 생성자를 통한 상속 - 부모 생성자 실행

- Person class의 내용을 prototype을 사용하여 구현하면

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

Person.prototype.sum = function(){
    return this.first + this.second;
}

function PersonPlus(name, first, second, third){
    Person.call(this,name,first,second);
    this.third = third;
}
PersonPlus.prototype.avg = function(){
    return (this.first+this.second+this.third)/3;
}

var kim = new PersonPlus('kim', 10, 20, 30);
console.log("kim.sum()", kim.sum());
console.log("kim.avg()", kim.avg());
```

PersonPlus가 Person을 상속하도록 하기 위해서 call 메소드를 사용하였습니다. Person의 this를 call를 통해 PersonPlus로 만들어지는 객체로 지정하여 부모의 생성자를 호출하였습니다. 하지만 아직 Person의 sum이라는 메소드는 PersonPlus에 상속되지 않았습니다.

13) 생성자를 통한 상속 - 부모와 연결하기

- 아직 Person과 PersonPlus가 아무런 연관이 없기 때문에 PersonPlus에서 Person에 소속되어 있는 sum 메소드를 호출하면 오류가 나게 됩니다. Person과 Person의 prototype 그리고 PersonPlus와 PersonPlus의 프로토타입 객체는 서로를 참조하고 있습니다. 그리고 PersonPlus를 통해 생성된 kim이라는 객체의 __proto__는 자신을 생성한 생성자의 prototype 객체인 PersonPlus 객체의 prototype 객체를 가리키고 있습니다. 이때 만약 kim에서 avg라는 함수를 호출하면 kim이라는 객체에 avg라는 프로퍼티가 없기 때문에 kim 객체의 __proto__가 가리키는 PersonPlus에서 avg라는 프로퍼티를 찾아서 실행하게 됩니다. 이번에는 kim이라는 객체에서 sum이라는 함수를 호출해보겠습니다. kim 객체 안에 sum이라는 프로퍼티가 없기 때문에 __proto__를 따라서 PersonPlus의 prototype 객체를 확인해보지만 역시나 sum이라는 프로퍼티가 존재 하지 않기 때문에 에러가 발생하게 됩니다. 따라서 우리는 PersonPlus의 porotype에 찾는 프로퍼티가 없을때는 Person의 prototype 객체를 확인하도록 연결해줘야합니다. 따라서 PersonPlus의 porotype의 __prototype__이 Person의 prototype 객체를 가리키도록 하면 됩니다.

```
function Person(name,first,second){
    this.name = name;
    this.first = first;
    this.second = second;
```

```
}

Person.prototype.sum = function(){
    return this.first + this.second;
}

function PersonPlus(name, first, second, third){
    Person.call(this,name,first,second);
    this.third = third;
}

PersonPlus.prototype.__proto__ = Person.prototype;

PersonPlus.prototype.avg = function(){
    return (this.first+this.second+this.third)/3;
}

var kim = new PersonPlus('kim', 10, 20, 30);
console.log("kim.sum()", kim.sum());
console.log("kim.avg()", kim.avg());
```

하지만 __proto__는 표준이 아니기 때문에 많은 예제에서는 Object.create()를 사용합니다. Object.create를 이용해 Person.prototype을 __proto__로 하는 새로운 객체를 생성한 후 PersonPlus의 prototype으로 지정합니다.

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

Person.prototype.sum = function(){
    return this.first + this.second;
}

function PersonPlus(name, first, second, third){
    Person.call(this,name,first,second);
    this.third = third;
}

//PersonPlus.prototype.__proto__ = Person.prototype;
PersonPlus.prototype = Object.create(Person.prototype);

PersonPlus.prototype.avg = function(){
    return (this.first+this.second+this.third)/3;
}
```

```
var kim = new PersonPlus('kim', 10, 20, 30);
console.log("kim.sum()", kim.sum());
console.log("kim.avg()", kim.avg());
console.log("kim.constructor",kim.constructor);
```
이때 constructor를 출력해보면 personPlus가 아닌 Person으로 나오는 것을 확인할 수 있습니다.

14) 생성자를 통한 상속 - constructor 속성은 무엇인가?

- Person은 __proto__라는 프로퍼티를 통해서 Person의 prototype 객체를 참조하고 Person의 prototype 객체는 constructor라는 프로퍼티를 통해서 Person 객체를 참조합니다. 만약 Person이라는 생성자를 통해서 생성된 kim이라는 객체에서 constructor를 호출한다면 kim에는 constructor라는 프로퍼티가 없기 때문에 kim의 __proto__가 가리키는 Person의 prototype 객체의 constructor가 가리키는 Person이 리턴되게 됩니다. 즉 자바스크립트에서 constructor라는 프로퍼티는 여러 의미로 사용되는데 그 중에 하나는 어떠한 객체가 누구로부터를 만들어졌는지를 알려주는 것입니다. 또 new 키워드와 함께 constructor()를 실행하면 constructor를 몰라도 새로운 객체를 생성할 수 있게 됩니다.

```
d = new Date()
d2 = new d.constructor()
// d2 = new Date()
```

15) 생성자를 통한 상속 - constructor 속성 바로잡기

- 이전에 PersonPlus로 생성된 kim 객체의 constructor가 Person으로 나오는 문제가 있었습니다. 이유는 PersonPlus의 prototype을 Person.prototype을 __proto__로 갖는 새로운 객체를 만들어 대체 했기 때문에 이전에 PersonPlus를 constructor로 갖는 PersonPlus의 prototype이 지워졌기 때문입니다. 따라서 PersonPlus.protype의 consturctor를 PersonPlus로 지정해주면 됩니다.

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

Person.prototype.sum = function(){
    return this.first + this.second;
}

function PersonPlus(name, first, second, third){
    Person.call(this,name,first,second);
    this.third = third;
}

//PersonPlus.prototype.__proto__ = Person.prototype;
PersonPlus.prototype = Object.create(Person.prototype);
PersonPlus.prototype.constructor = PersonPlus;

PersonPlus.prototype.avg = function(){
```

```
    return (this.first+this.second+this.third)/3;
}

var kim = new PersonPlus('kim', 10, 20, 30);
console.log("kim.sum()", kim.sum());
console.log("kim.avg()", kim.avg());
console.log("kim.constructor",kim.constructor);
```

사실 이러한 코드를 실제로 사용하는 것보다는 class를 사용하는 편이 더 깔끔하고 직관적입니다. 하지만 class의 내부에 이러한 방식으로 동작된다는 것을 알고 이해했다면 자바스크립트의 상당히 중요하고 어려운 부분을 이해하게 되었다는 뜻과도 같습니다.

7. Javascript 활용

1) 파일로 쪼개서 정리 정돈하기

- 파일을 이용해 코드 정리하기 : 한 파일에 있는 부분을 여러 페이지에도 복사/붙여넣기를 한다고 할 때 태그 외에도 만들어 놓은 Javascript 코드까지 복사하고 붙여넣어야 합니다. 하지만 붙여넣어야 할 페이지가 매우 많으면, 코드를 넣기도 힘들고, 수정하기도 힘들어집니다. 이러한 상황에서 사용할 수 있는 방법이 바로 파일로 쪼개는 것입니다. 새로운 js 파일을 만들어 봅시다. colors.js라는 파일 이름으로 만들었다고 할 때 그 파일에 모든 페이지에서 공통적으로 사용되는 Javascript 코드를 넣습니다. 그렇다면 원래 Javascript 코드가 있었던 <script> 태그는 src 속성에 이 파일 이름을 넣어서 바꾸면 됩니다.

```
<script src='colors.js'></script>
```

이렇게 바꾼 후 페이지를 실행해보면, 웹 브라우저가 colors.js 파일을 자동으로 가져와서 원래와 똑같이 작동하는 것을 볼 수 있습니다.

- 파일 이용의 장점 : 짧은 script 코드를 필요한 페이지에 붙여넣으면 이 Javascript 코드를 한 번에 관리할 수 있는 것입니다. 즉, 코드를 재사용할 수 있고, 동시에 코드를 수정할 수 있어서 유지보수가 편리해집니다. 코드가 명확해지고 가독성이 좋아진다는 장점도 있습니다. 이렇게 파일을 분리하면 캐시의 입장에서도 장점을 가집니다. colors.js라는 파일을 한 번 다운로드 해서 캐시에 저장해두면 다운로드 없이 사용할 수 있기 때문에 더 빨리 페이지를 표시할 수 있는 것입니다.

2) 라이브러리와 프레임워크

- 소프트웨어의 사회성 : 소프트웨어는 혼자서 만드는 것이 아닙니다. 다른 사람들이 이미 잘 만들어 놓은 코드를 가지고 빠르게 조립해서 우리가 원하는 코드를 만들 수 있다는 것입니다.

다른 사람과 코드를 통해서 협력하는 모델에는 두 가지가 있습니다. 라이브러리는 프로그램에 필요한 부품이 되는 소프트웨어가 정리되어 있는 것입니다. 프레임워크는 만들고자 하는 프로그램의 종류에 따라서 공통적인 부분을 미리 만들어 놓는 것입니다. 필요한 부분만 약간 수정해서 사용할 수 있는 것이죠. 즉, 라이브러리는 우리가 필요한 부분을 가져와서 사용하는 것이라면, 프레임워크는 직접 프레임워크 안으로 들어가서 디테일을 수정해서 사용하는 것이라고 생각하면 됩니다.

- jQuery 라이브러리 : jQuery 라이브러리를 사용하면 생산성을 높일 수 있습니다. 인터넷에서 jQuery를 다운로드 하는 방법도 있고, CDN이라는 방법을 사용해도됩니다. CDN을 사용하면 코드를 한 줄 추가하는 것만으로도 라이브러리를 가져와서 사용할 수 있습니다.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

이 한 줄을 <head>에 추가하면 된다는 것입니다. jQuery를 사용하면 반복문을 사용하지 않고도 모든 태그를 한 번에 처리할 수 있습니다.

```
$('a').css("color","powderblue");
```

반복문을 사용하지 않고 이 한 줄만으로 모든 a 태그의 색깔을 powderblue로 바꿀 수 있는 것입니다. 하지만 jQuery는 새로운 언어가 아닙니다. 직접 함수를 만드는 대신 이러한 일을 하는 함수가 jQuery 안에 미리 만들어져 있는 것입니다. 이렇게 적절한 라이브러리를 잘 사용하면 효율적으로 코딩을 할 수 있게 됩니다.

3) UI vs API

- UI와 API : UI는 User Interface의 약자입니다. API는 Application Programming Interface의 약자입니다. 어떤 페이지에 버튼이 하나 있고, 이 버튼을 누르면 경고창이 뜨게 만들었다고 해 봅시다. 이 버튼을 사용하는 것은 웹페이지의 사용자죠. 사용자들이 시스템을 제어하기 위해서 조작하는 장치를 UI라고 부릅니다. 경고창의 텍스트나, 경고창이 뜨는 타이밍은 우리가 만든 것이지만, 우리는 경고창을 실제로 띄우기 위해서는 alert라는 함수를 사용했습니다. alert라는 이 함수는 웹 브라우저를 만든 사람들이 미리 만들어둔 것입니다. 그리고 이 alert라는 함수를 호출해서 조작하는 것입니다. 이렇게 프로그래머들이 사용하는 조작 장치들을 API라고 부릅니다. 즉, alert는 API인 것입니다. 이러한 UI와 API라는 개념은 Javascript 뿐만 아니라 모든 프로그래밍 언어에 적용되는 개념입니다. 우리는 API를 응용하고 결합해서 새로운 프로그램을 만들 수 있는 것입니다.