<혼자 공부하는 자바스크립트>

Chapter 01. 자바스크립트 개요와 개발환경 설정

01-1 자바스크립트의 활용

자바스크립트(Javascript) : 웹 브라우저에서 사용하는 프로그래밍 언어

자바스크립트로 할 수 있는 것들?

- 웹 클라이언트 애플리케이션 개발
- 웹 서버 애플리케이션 개발
- 모바일 애플리케이션 개발 ; 네이티브 앱, 모바일 웹 앱, 하이브리드 앱
- 데스크톱 애플리케이션 개발
- 데이터베이스 관리

01-2 개발환경 설치와 코드 실행

코드 실행기 : 구글 크롬

텍스트 에디터 : 비주얼 스튜디오 코드

코드 실행하는 방법?

- 구글 크롬 콘솔에서 실행하기

  간단하게 테스트해야 하는 상황에 주로 사용

  기존 웹 페이지의 코드와 충돌을 방지하기 위해 'about:blank'에서 하기

- 파일 만들고 저장해 실행하기

  (1) [파일] -> [새 파일(Ctrl+N)]을 선택

  (2) [파일] -> [저장(Ctrl+S)]를 선택

  (3) "파일명.html" 라는 형태로 저장

  (4) html:5 입력하고 엔터 또는 탭

  (5) head 태그 안에 script 태그 생성

  (6) script 태그 안에 코드를 입력

  (7) 코드를 모두 입력했다면 저장

  (8) 크롬에 드래그앤드롭 해서 실행

오류 확인

(1) Reference Error(예외 처리) : 단어 오탈자가 발생하면 뜬다.

(2) Syntax Error(구문 오류) : 일반적으로 기호에서 오탈자가 발생하면 뜬다.

*에디터

- [파일] -> [기본설정] -> [색테마(Ctrl+K Ctrl+T]
- Shift + 방향키_위/아래 : 위/아래 줄 선택
- Tab / Shift+Tab으로 들여쓰기 추가 제거
- [파일] -> [기본설정] -> [설정] -> Tab Size를 2로 변경
- 멀티 커서 : 마우스 사용 형태
- Alt + 마우스 클릭
- [파일] -> [기본설정] -> [설정] -> Font Family를 d2coding

01-3 알아두어야 할 기본 용어

- 표현식(expression) : 값을 만들어내는 간단한 코드

10-> 표현식

10+10 -> 20을 만들어내는 표현식

"안녕하세요" -> 표현식

주의! 표현식은 값의 관점에서 보는 것, 문장은 실행의 관점에서 보는 것이라 두 가지 속성을 동시에 가질 수도 있습니다. (표현식이면서 문장이다.)

표현식 하나를 나타내는 문장을 나중에 프로그래밍 언어 고급 또는 컴파일러를 배우면 표현식 문장, 식문이라고 부른다.

자연 언어의 문장은 sentence, 프로그래밍의 문장은 statement

- 문장(statement) : 하나 이상의 표현식이 모인 것으로 문장 끝에는 세미콜론(;) 또는 줄바꿈을 넣는다. 문장의 종결이 이뤄지면 그때까지가 한 문장, 명사(표현식) 하나도 종결만 시키면 문장이 된다.

할당문 -> 문 들어가니까 문장 -> assignment statement

if문 or if 조건문 -> 문 들어가니까 문장 -> if statement or if conditional statement

for문 or for 반복문 -> 문 들어가니까 문장 -> for statement or looping statement

- 키워드(keyword) : 자바스크립트가 처음 만들어질 때 정해놓은 특별한 의미가 있는 단어

await, break, case, catch, class, const, continue, debugger, default, delete, do, else, export, extends, finally, for, function, if, impoert, in, instanxeof, new, return, super, switch, this, throw, try, typeof, var, void, while, with, yield, let, static, true, false, null, as, from, get, of, set, target

- 식별자(identifier) : 프로그래밍 언어에서 이름을 붙일 때 사용하는 단어, 코드 실행기가 문법적으로 완전 특별 취급한다.

키워드 사용 X, 숫자로 시작 X, 특수문자는 _와 $만 허용, 공백 문자 포함 X

대문자로 시작 + 소문자 -> 클래스

식별자 뒤에 괄호 X + 단독으로 사용 -> 변수

식별자 뒤에 괄호 X + 다른 식별자와 사용 -> 속성

식별자 뒤에 괄호 + 단독으로 사용 -> 함수

식별자 뒤에 괄호 + 다른 식별자와 사용 -> 메소드 ; 다른 식별자와 사용은 "앞에 점 찍고 사용"으로 생각

주황색 + ture/false -> 키워드, 나머지 이름은 전부 식별자

['user strict' 등의 따옴표로 감싸진건 식별자가 아님]

- 주석(comment) : 프로그램 코드를 사람에게 설명하기 위한 목적을 가진 코드, 코드 실행기(기계)는 필요 없으므로 무시

HTML 주석 <!--주석 내용-->

자바스크립트 주석

// 한 줄 주석

/* 여러 줄 주석 */

Chapter 02. 자료와 변수

02-1 기본 자료형

자료형 : 프로그래밍에서 프로그램이 처리할 수 있는 것을 자료라 하고, 자료 형태에 따라 나눠 놓은 것

함수와 메소드는 모두 '특정 기능을 동작시키도록 작성' 된 '코드의 집합'

메소드는 클래스가 가지고 있는 함수이며, 함수는 메소드를 포괄하는 의미

- 문자열(string) 자료형 : 문자들을 작은 따옴표 또는 큰 따옴표로 감싸면 문자열이 만들어진다.

이스케이프 문자(=역슬래시)는 엔터 위에 있는 원 기호(₩ or \)

\n (줄바꿈), \t (탭 문자), \\ (역슬래시 자체)

문자열에 적용할 수 있는 처리?

(1) 문자열 연결 연산 : 문자열 + 문자열

(2) 문자 선택 연산 : 문자열[인덱스] -> 문자 하나

(3) 문자열의 길이 : 문자열.Length -> 문자 개수

오류 확인

(1) Uncaught SyntaxError: Unexpected identifier(구문 오류) : 식별자 주변에 잘못된 코드가 있다.

- 숫자(number) 자료형 : 소수점이 있는 숫자와 없는 숫자를 모두 같은 자료형으로 인식

+ (더하기 연산자), - (빼기 연산자), * (곱하기 연산자), / (나누기 연산자), % (나머지 연산자)

나머지 연산자는 좌변을 우변으로 나눈 나머지를 출력

! 컴퓨터는 한정된 자원으로만 자료를 표현하므로 그 크기를 넘는 개념 또는 무한한 개념을 표현하지 못한다. (이진수 실수 표현)

- 불(boolean) 자료형 : 참과 거짓 값을 표현, true와 false 2가지

불을 만드는 법? true 또는 false를 그대로 입력 -> 비교 연산자를 사용

비교 연산자 : === (양쪽이 같다), !== (양쪽이 다르다), > (왼쪽이 더 크다), < (오른쪽이 더 크다),

>= (왼쪽이 더 크거나 같다), <= (오른쪽이 더 크거나 같다)

(1) 논리 부정 연산자 : !불

단항 연산자; 피연산자가 하나, 이항 연산자; 피연산자가 두 개

(2) 논리합/곱 연산자 : ||(또는) / &&(그리고)

논리합 연산자 = 불 || 불 -> 적어도 하나만 true면 전체 값이 true

논리곱 연산자 = 불 && 불 -> 양쪽 모든 것이 true여야 전체 값이 true

- 자료형 검사 : typeof(자료) 입력하면 자료형이 문자열 또는 숫자 또는 불로 나온다. typeof는 연산자로 사용되는 키워드

템플릿 문자열 : `는 backtick, grave accent라 해서 역틱, 백틱, 억음부호, 그레이브 액센트라고 부르는 기호(Tap키 위에 키) 문자열 내부에 '${...}'기호를 사용해 표현식을 넣으면 표현식이 문자열 안에서 계산된다.

02-2 상수와 변수

*식별자를 상수/변수로 사용 -> 선언 또는 정의

'식별자 = '자료' ' -> 할당한다

처음으로 값을 할당하는 것 -> 초기화

- 상수(constant) : '항상 같은 수' 라는 의미로 값에 이름을 한 번 붙이면 값을 수정할 수 없다.

상수 만드는 법?

(1) const 식별자 = 자료 -> 고정된 형태로 다르게 쓰면 오류

Identifier has already declared(구문 오류) : 한 파일에서 같은 이름으로 상수를 선언할 때 발생

Missing initializer in const declaration(구문 오류) : 상수 선언할 때 값을 지정해주지 않으면 발생

Assignment to constant variable(예외 처리) : 한 번 선언된 상수의 자료는 변경할 수 없다.

오류 해결 방법? 다른 이름 사용, (이전 내용이 필요 없으면) 새로고침

- 변수(variable) : '변할 수 있는 수' 로 값을 수정할 수 있다.

변수 만드는 법?
(1) let 식별자
(2) let 식별자 = 자료
변수에 값 넣기 : 변수 = 자료
변수 사용하기 : 변수
Identifier has already been declared(구문 오류) : 한 파일에서 같은 이름으로 변수를 선언할 때 발생
변수에 적용할 수 있는 연산자
(1) 복합 대입 연산자
+= (기존 변수의 값에 값을 더함), -= (기존 변수의 값에 값을 뺌), *= (기존의 변수의 값에 값을 곱함),
/= (기존 변수의 값에 값을 나눔), %= (기존 변수의 값에 나머지를 구함)
(2) 증감 연산자; (전위 > 후위)
변수++ (기존의 변수의 값에 1을 더합니다(후위)), ++변수 (기존의 변수의 값에 1을 더합니다(전위)),
변수-- (기존의 변수의 값에 1을 뺍니다(후위)), --변수 (기존의 변수의 값에 1을 뺍니다(전위))
- undefined 자료형
(1) 상수와 변수로 선언하지 않은 식별자
typeof(a) -> "undefined"
(2) 값이 없는 변수
let b -> undefined
typeof(b) -> "undefined"

02-3 자료형 변환

- 문자열 입력 : prompt(메시지 문자열, 기본 입력 문자열)

```
<script>
  // 상수를 선언
  const input = prompt( 'message' , '_default' )

  // 출력
  alert(input)
</script>
```

-> 사용자에게 입력을 요구하는 창이 나타난다. 입력 양식에 글자를 입력하고 [확인]을 클릭하면, 입력한 글자가 경고창에 출력되고, prompt() 함수는 입력한 문자열을 input에 저장한다. 예를들어 '안녕하세요'를 입력했을 때, const input = '안녕하세요'로 변환된다. 즉, input에 저장된 문자열 '안녕하세요'를 출력(리턴)한다.

- 불 입력 : confirm(메시지 문자열)

```
<script>
  // 상수를 선언
  const input = confirm( '수락하시겠습니까?' )

  // 출력
  alert(input)
</script>
```

-> 사용자에게 확인을 요구하는 메시지 창이 나타난다. [확인]을 클릭하면 true를 리턴하고, [취소]를 클릭하면 false를 리턴한다. input에 불 자료형이 들어가고 곧바로 input에 저장된 값을 출력한다.

- 숫자 자료형으로 변환하기 : Number(자료)

Number("273") -> 273 typeof(Number("273")) -> "number"

Number(“$273”) -> NaN typeof(Number(“$273”)) -> “number”
NaN은 ‘Not a Number’ 로 숫자가 아니다지만, 자료형은 숫자가 맞다.
Number(true) -> 1 Number(false) -> 0

- 문자열 자료형으로 변환하기 : String(자료)

String(52.273) -> “52.273”
String(true) -> “true” String(false) -> “false”

- 불 자료형으로 변환하기 : Boolean(자료)

Boolean(0) -> false
Boolean(NaN) -> false
Boolean(“”) -> false
Boolean(null) -> false
let 변수 -> undefined Boolean(변수) -> false
!!273 -> true !!0 -> false
!! ‘안녕하세요’ -> true !! ‘’ -> false

- inch를 cm 단위로 변경하기

```
<script>
  // 프로그램(program = pro[미리] + gram[작성된 것])
  // 숫자를 입력
  const rawInput = prompt( ‘inch 단위의 숫자를 입력해주세요.’ )

  // 입력받은 데이터를 숫자형으로 변경하고 cm 단위로 변경
  // 1inch = 2.54cm
  const inch = Number(rawInput)
  const cm = inch * 2.54

  // 출력 cm 단위의 숫자
  alert(‘${inch}inch는 ${cm}cm입니다.’)
</script>
```

-> inch 단위의 숫자를 입력해주세요. 입력> 10
-> 10inch는 25.4cm

Chapter 03. 조건문

03-1. if 조건문

제어문(control statement) : 코드의 실행 흐름과 직접적인 관련이 있는 문장

if 조건문 : 조건에 따라서 코드를 실행하거나 실행하지 않을 때 사용하는 구문

- 이때 조건은 불 자료형을 의미한다.
- 비교 연산자와 논리 연산자를 활용해 조건을 만들고, 이 조건을 사용해 조건 분기
- if (불 값이 나오는 표현식) {
    불 값이 참일 때 실행할 문장
  }

if else 조건문 : if 조건문을 2번 사용하고, else로 조건문을 사용할 수 있도록 서로 반대되는 상황을 표현하는 구문을 제공

- if (불 값이 나오는 표현식) {
  불 값이 참일 때 실행할 문장
  } else {
  불 값이 거짓일 때 실행할 문장
  }

중첩 조건문 : 조건문 안에 조건문을 중첩해 사용하는 것

- if (불 값이 나오는 표현식1) {
    if (불 값이 나오는 표현식2) {
      표현식 2가 참일 때 실행할 문장
    } else {
      표현식 2가 거짓일 때 실행할 문장
    }
  } else {
    if (불 값이 나오는 표현식3) {
      표현식 3이 참일 때 실행할 문장
    } else {
      표현식 3이 거짓일 때 실행할 문장
   }
  }

if else if 조건문 : if 조건문은 조건이 한 문장이라면 중괄호 생략 가능, 중첩 조건문에서 중괄호를 생략한 형태, 겹치지 않는 3가지 이상의 조건으로 나눌 때 사용

- if (불 표현식) {
    문장
   } else if (불 표현식) {
    문장
   } else if (불 표현식) {
    문장
   } else {
    문장
   }

03-2. switch 조건문과 짧은 조건문

switch 조건문 : 괄호 안에는 비교할 값 입력, 만약 입력한 표현식과 case 키워드 옆의 표현식이 같다면 case 키워드

바로 다음에 오는 문장을 실행, 중괄호는 사용하지 않아도 됨.

- switch (자료) {
  case 조건A:
   break
  case 조건B:
   break
  default: -> 생략 가능
   break
 }

- break 키워드 : switch 조건문이나 반복문을 빠져나가기 위해 사용, break 키워드를 만나면 break 키워드를 감싼 switch 조건문이나 반복문을 완전히 빠져나간다.

조건부 연산자 : 조건문과 비슷한 역할, 자바스크립트에서 유일하게 항을 3개 갖는 연산자로 '삼항 연산자'라고도 함

- 불 표현식 ? 참일 때의 결과 : 거짓일 때의 결과

짧은 조건문 : 논리 연산자의 특성을 조건문으로 사용

- 논리합 연산자를 사용한 짧은 조건문
  true || 000
  불 표현식 || 불 표현식이 거짓일 때 실행할 문장
- 논리곱 연산자를 사용한 짧은 조건문
  false && 000
  결과가 거짓인 불 표현식 && 불 표현식이 참일 때 실행할 문장

*날짜에 대한 궁금증! 왜 Month는 0부터 시작하는데 Date는 1부터 시작하느냐, 개발자가 그렇게 만들었기 때문입니다. 프로그래밍 언어마다 다 다르니 콘솔 등에서 한두 번 입력해보고 어떤 값이 나오는지 확인하고 사용하세요.

자바스크립트 기준!

- FullYear : 올해(ex: 2021)
- Month : 월(ex: 0~11) -> 1월이 0입니다.
- Date : 일(ex: 1~31) -> 1일이 1입니다.
- Hours : 시간(ex: 0~23)

등등이 있습니다.

Chapter 04. 반복문

04-1. if 배열

배열(array) : 여러 자료를 묶어서 활용할 수 있는 특수한 자료, 여러 개의 변수를 한 번에 선언해 다룰 수 있는 자료형 즉, 여러 개의 값을 모아놓은 것

대괄호[…]를 사용해 생성하고 내부의 값을 쉼표(,)로 구분해 입력

- 요소(element) : 배열 내부에 들어 있는 값

  [요소, 요소, 요소, …, 요소]

- 인덱스(index) : 요소의 순서, 가장 앞에 있는 요소를 0번째로 표현

  배열[인덱스]

- length 속성 : 배열 내부에 들어 있는 요소의 개수를 확인

  배열.length

- push() 메소드 : 배열 뒷부분에 요소 추가

  배열.push(요소)

- 인덱스를 사용해 배열 뒷부분에 요소 추가

ex) > const fruitsA = ['사과', '배', '바나나']

Undefined

> fruitsA[3] = '귤'

"귤"

> fruitsA

(4) ["사과", "배", "바나나", "귤"]

- 배열 요소 제거하기

첫째, 인덱스를 기반으로 제거하는 경우

splice() 메소드 '접합'은 일부를 제거한 뒤 붙이는 것뿐만 아니라, 중간에 다른 요소를 넣고 붙이는 것도 접합

배열.splice(인덱스, 제거할 요소의 개수)

> const itmesA = ['사과', '배', '바나나']

Undefined

> itemsA.splice(2, 1)

["바나나"]

> itemsA

(2) ["사과", "배"]

둘째, 값을 기반으로 제거하는 경우

indexOf() 메소드를 사용해 배열 내부에서 특정 값의 위치를 찾고, 위치를 추출한 뒤 splice() 메소드를 사용해 제거

const 인덱스 = 배열.indexOf(요소)

배열.splice(인덱스, 1)

! indexOf() 메소드는 배열 내부에 요소가 있을 경우 인덱스를 리턴, 배열 내부에 요소가 없을 때는 -1을 리턴

> const itmesB = ['사과', '배', '바나나']

Undefined

> const index = itemsB.indexOf('바나나')

Undefined

> index

2

> itemsB.splice(2, 1)

["바나나"]

> itemsB
(2) [“사과” , “배”]
> itemsB.indexOf(‘바나나’)
-1

*문자열의 indexOf() 메소드 : 문자열 내부에서 특정 문자열의 위치를 찾을 수 있다. 다음 코드는 ‘동해물과 백두산이 마르고 닳도록’ 이라는 문자열 내부에서 ‘백두산’ 의 위치를 찾습니다. 백두산의 앞 글자인 ‘백’ 의 위치가 5번째 인덱스에 있으므로 5를 출력

*배열 내부에서 특정 값을 가진 요소 모두 제거 : indexOf() 메소드와 splice() 메소드는 배열 내부 요소를 하나만 제거할 수 있고, 배열 내부에서 특정 값을 가진 요소를 모두 제거하고 싶을 때는 반복문을 사용하거나 filter() 메소드를 사용해야 한다. 일반적으로 filter() 메소드를 많이 사용

> const itemsE = [‘사괴’ , ‘배’ , ‘바나나’ , ‘귤’ , ‘귤’]
undefined
> itemsE.filter((item) => item !== ‘귤’)
(3) [“사과” , “배” , “바나나”]

- 배열의 특정 위치에 요소 추가하기

배열.splice(인덱스, 0, 요소)

splice() 메소드의 2번째 매개변수에 0을 입력하면 splice() 메소드는 아무 것도 제거하지 않으며, 3번째 매개변수에 추가하고 싶은 요소를 입력

* 기본 자료형(숫자, 문자열, 불) < 복합 자료형(배열, 함수, 객체)

어떤 값이 메모리 어디 있는지를 알면 프로그램의 값을 조작할 수 있다.

스택 : 스택스택 쌓는 곳으로 기본 자료형과 주소 등을 저장하는 메모리 공간

힙 : 힙힙 던져서 쌓는 곳으로 복합 자료형을 저장하는 메모리 공간

주소 : 저장된 자료의 위치

레퍼런스한다. : 스택의 주소가 힙의 자료를 가리키는 것

레퍼런스 변수 : 스택에 저장된 것중에 주소가 저장된 변수

* 자료의 비파괴와 파괴

자료 처리를 위해 다양한 연산자, 함수, 메소드를 제공하는데 자료 처리 연산자, 함수, 메소드는 크게 비파괴적 처리와 파괴적 처리로 구분할 수 있다.

- 비파괴적 처리 : 처리 후에 원본 내용이 변경되지 않는 처리로, 메모리가 여유로운 현대 프로그래밍 언어와 라이브러리는 자료 보호를 위해 대부분 비파괴적 처리를 한다.

원본도, 새로운 생성본도 둘 다 메모리에 존재해야 한다.

ex) 필요 없는 출력은 생략.

> const a = ‘안녕’
> const b = ‘하세요’
> const c = a + b
> c
“안녕하세요”
> a
“안녕”
> b
“하세요”

- 파괴적 처리 : 처리 후에 원본 내용이 변경되는 처리로, 1995년 경에는 컴퓨터 메모리가 많이 부족해 커질 가능성이 많은 배열을 복제해서 생성하는 것이 컴퓨터에게 굉장한 부담이 되기에, 복제해서 생성하는 것을 포기하고 기존에 있던 배열에 값을 그냥 직접적으로 수정하는 파괴적 방법을 사용했다. 프로그래밍 언어와 라이브러리들은 최대한 메모리를 절약

해서 사용하는 방식으로 설계, 배열과 같이 거대해질 수 있는 자료는 메모리를 절약할 수 있게 대부분 파괴적 처리로 이뤄졌습니다. 메모리 절약할 수 있지만, 원본이 사라지기 때문에 위험할 수 있습니다.

ex) 필요 없는 출력은 생략

> const array = ["사과" , "배" , "바나나"]

> array.push("귤")

4

> array

(4) ["사과" , "배" , "바나나" , "귤"]

루비 등의 프로그래밍 언어는 메서드 이름을 보면 비파괴적인지, 파괴적인지 알 수는 있지만, 자바스크립트는 코드를 여러 번 실행해보면서 외우는 수밖에 없습니다. 그래서 개념을 기억해뒀다가 그 쓰임을 익히는 것이 좋은 방법입니다.

* const의 제한 : const -> 스택에 있는 값 변경할 때 오류 -> 힙에 있는 레퍼런스된 복합 자료형을 조작하는 것에는 문제가 없다. const가 있어도 배열 등의 처리는 가능하다.

04-2. 반복문

반복문 : 컴퓨터에게 100번, 1000번 혹은 무한히 반복하고 싶을 때 코드를 계속 붙여넣는 것은 무리다. 이때 활용하는 것이 반복문입니다.

- for in 반복문 : 배열과 함께 사용할 수 있는 반복문으로, 배열 요소를 하나하나 꺼내서 특정 문장을 실행할 때 사용

```
for (const 반복 변수 in 배열 또는 객체) {
  문장
}
```

for 반복문의 반복변수에는 요소의 인덱스들이 들어온다. 이를 활용해 배열 요소에 접근할 수 있다.

for 입력 -> forin 코드 블록 -> Enter 또는 Tab

```
for (const key in object) {
  if (object.hasOwnProperty(key)) {
    const element = object[key];

  }
}
```

- for of 반복문 : 요소의 값을 반복할 때 안정적으로 사용할 수 있다.

for in 반복문은 반복 변수에 인덱스가 들어가고 반복문 내부에 요소를 사용하려면 fruits[i]와 같은 형태로 사용하며, 안정성을 위해 몇 가지 코드를 더 추가해야하는데 그래서 등장한 것이다.

```
for (const 반복 변수 of 배열 또는 객체) {
  문장
}
```

for 입력 -> forof 코드 블록 -> Enter 또는 Tab

```
for ( const iterator of object) {

}
```

- for 반복문 : 특정 횟수만큼 반복하고 싶을 때 사용하는 범용적인 반복문

```
for (let i = 0; i < 반복 횟수; i++) {
  문장
}
```

다른 반복문과 다르게 반복 변수를 let 키워드로 선언

for 입력 -> for 코드 블록 -> Enter 또는 Tab

```
for (let index = 0; index < array.length; index++) {
  const element = array[index];

}
```

for 반복문은 배열과 조합할 수 있다. 보통 배열의 length 속성만큼 반복을 돌리는 형태로 사용한다. 또한, 빠른 속도로 많은 양을 반복 작업하기 때문에 사용 범위가 넓습니다.

- while 반복문 : if 조건문과 형태가 매우 비슷한 반복문으로, if 조건문과 다른 점은 무장을 한 번만 실행하고 끝나는 것이 아니라 불 표현식이 true면 계속해서 문장을 실행한다는 것으로, 조건이 변하지 않는다면 무한히 반복 실행하므로 조건을 거짓으로 만들 수 있는 내용이 문장에 포함되어 있어야 한다. 반복문이 무한 반복되는 것을 무한 루프하고 한다.

```
while (불 표현식) {
  문장
}
```

반복문의 조건식에 confirm() 함수를 넣어 함수를 입력하면 사용자에게 확인을 받는 대화상자가 실행된다. [확인] -> true가 되어 반복문을 계속해서 반복, [취소] -> false로 바뀌어 반복을 종료

for 반복문은 횟수를 기준으로 반복할 때는 코드를 간결하게 구현할 수 있는 것을 사용, while 반복문은 조건에 큰 비중이 있을 때 '특정 시간 동안 어떤 데이터를 받을 때까지', '배열에서 어떠한 요소가 완전히 제거될 때까지' 등 조건을 기반으로 사용하는 반복문에 사용한다.

* 반드시 무한 반복을 벗어나게 코드 구현하기 : 다른 프로그래밍 언어에서는 데이터를 전달받을 때까지 기다린다와 같은 목적으로 무한 반복문을 사용하기도 한다. 하지만, 자바스크립트는 무한 반복을 사용하면 페이지 전체가 먹통이 되는 문제가 발생하므로 break 구문 등을 활용해 반드시 무한 반복을 벗어나도록 코드를 구현해야 한다.

- break 키워드 : switch 조건문에서 언급했듯이 switch 조건문이나 반복문을 벗어날 때 사용, while 반복문은 조건이 항상 참이므로 무한 반복, 이러한 무한 루프는 break 키워드를 사용해야 벗어날 수 있다.

```
while (true) {

} break
```

- continue 키워드 : 반복문 안의 반복 작업을 멈추고 반복문의 처음으로 돌아가 다음 반복 작업을 진행한다.

! break 키워드나 continue 키워드를 적당히 사용하면 코드가 간결해 보이지만, 그런 적당한 경우는 드물다. 반복문의 조건식을 적절하게 만들면 필요 없는 구문이기도 한다. 프로그래밍을 처음 배울 때는 조건식 만드는게 익숙하지 않아 break 키워드나 continue 키워드가 필요 없는 부분에도 무리하게 사용하는데, 최대한 자제할 수 있도록 주의

Chapter 05. 함수

05-1. 함수의 기본 형태

함수? 코드의 집합으로 함수를 실행하면 여러 코드를 한 번에 묶어서 실행할 수 있으며, 필요할 때마다 호출해 반복적으로 사용할 수도 있다. 중괄호 {…} 내부에 코드에 넣기에 집합이라고 한다.

- 자료형 : function
- 호출 : f( ) { }

```
// 익명 함수
const f = funtion(매개변수, 매개변수) {
   return 리턴값
 }

// 선언적 함수
 funtion f(매개변수, 매개변수) {
   return 리턴값
 }
```

-> 초기에는 이 아래 형태를 더 많이 사용했고, 아래 형태가 일반적인 프로그래밍 언어에서 사용하는 방식이지만, 자바스크립트가 나름대로 독자적인 노선으로 프로그래밍 언어의 본질을 조금씩 발전시켜 나가자 현대 자바스크립트는 이 위를 사용한다.

- 함수 호출 : 함수를 사용하는 것으로, 함수를 호출
- 매개변수 : 함수를 호출할 때는 괄호 내부에 여러 가지 자료를 넣는 자료
- 리턴 값 : 함수를 호출해서 최종적으로 나오는 결과, 함수 본문에서 호출 위치로 나오는 것
- 점프 : 호출 위치에서 함수 본문으로 이동하는 것
- 만드는 방법 : 프로시저적 관점, 수학적 관점, 프로그래밍 관
- 함수를 사용하면 좋은 점

반복되는 코드를 한 번만 정의해놓고 필요할 때마다 호출하므로 반복 작업을 피할 수 있습니다.
긴 프로그램을 기능별로 나눠 여러 함수로 나눠 작성하면 모듈화로 전체 코드의 가독성이 좋아집니다.
기능별(함수별)로 수정이 가능하므로 유지보수가 쉽습니다.

익명 함수(anonymous function) : 함수를 출력했을 때 별다른 이름이 붙어있지 않은 함수, f( ) { }

선언적 함수 : 이름이 있는 함수를 많이 사용합니다.

```
 function 함수() {

 }
 let 함수 = function ( ) { };
```

매개변수 : 함수를 호출할 때 괄호 안에 적는 것, prompt() 함수를 사용할 때 매개변수로 message를 넣어야 한다. 그러면 prompt() 함수의 최종 결과는 문자열로 나옵니다. input

리턴값 : 함수의 최종 결과. output

```
 function 함수(매개변수, 매개변수, 매개변수) {
   문장
   문장
   return 리턴값
 }
```

나머지 매개변수(rest parameter) : funtion 함수 이름(. . .나머지 매개변수) { }, 매개변수 앞에 마침표 3개를 입력하면 매개변수들이 배열로 들어온다.

// 나머지 매개변수
const 함수 = function (. . .매개변수) {}
// -> 배열!
- 가변 매개변수 함수 : 호출할 때 매개변수의 개수가 고정적이지 않은 함수
- 나머지 매개변수와 일반 매개변수 조합 : funtion 함수 이름(매개변수, 매개변수, …나머지 매개변수) { }
- min(배열) 형대로 매개변수에 배열을 넣으면 배열 내부에서 최솟값을 찾아주는 함수
- min(숫자, 숫자, …) 형태로 매개변수를 넣으면 숫자들 중에서 최솟값을 찾아주는 함수
- typeof 연산자 : 매개변수로 들어온 자료형이 배열인지 숫자인지 확인할 수 있어야 한다. 자료형 typeof 연산자를 사용해서 쉽게 확인할 수 있다. 또한, 배열에 typeof 연산자를 사용하면 object(객체)라는 결과가 나온다. 일반적인 typeof 연산자로는 배열을 확인할 수 없다.
typrof(배열) == 'object'
Array.isArray() 메소드 : 정확하게 배열인지 확인하는 메소드
- 전개 연산자(spread operator) : 자바스크립트는 배열을 전개해서 함수의 매개변수로 전달해준다.
함수 이름(…배열)
// 전개 연산자
함수(. . .배열)
기본 매개변수 : 매개변수를 여러 번 반복해서 입력하는 것이 귀찮을 때 매개변수에 기본값을 지정
함수 이름(매개변수, 매개변수=기본값, 매개변수=기본값)
function sample(a=기본값, b) { }
일반 매개변수 위치에 값을 안 넣으면 undefinde가 들어간다.
* API(Application Programming Interface(약속) : 애플리케이션 프로그램을 만들 때의 약속
Web API, Win API, JavaScript API, Node API 등 종류가 거대하다.
alert()의 구현을 몰라도 추상화의 관점에서 옳기 때문에 써도 된다.
* 가독성 : 코드를 쉽게 읽고 이해하고, 안전하게 쓸 수 있는 속성
가독성이 중요한 이유 : 기업 입장의 비용 절감, 프로그램이 너무 복잡해져서, 지원 도구의 활용
현대적인 개발에서는 "단위 테스트(unit test)"가 중요해졌다; 코드를 작성하고 빠르게 문제가 있는지 없는지 테스트해주는 과정을 자동화한 것
* 함수 헤더(함수의 이름, 매개변수로 뭘 넣어야하는지)만 봐도 이걸 어떻게 사용할 수 있는지 알아야 한다는 암묵적 룰
잉 ITsmsep 기존의 자바스크립트에서 이렇게 가변 매개변수 함수를 만들면 이런 것이 불가능했습니다. 지금은 가변 매개변수 함수를 만들 수 있는 문법이 나왔고 + arguments를 사용하는 형태는 위험한 형태(뭐도 모르니까)이므로 사용하지 않는 것을 추천합니다!

05-2. 함수 고급

콜백(callback) 함수 : 매개변수로 전달하는 함수
- forEach() : 콜백 함수를 활용하는 가장 기본적인 함수, 메소드는 배열이 갖고 있는 함수(메소드)로써 단순하게 배열 내부의 요소를 사용해서 콜백 함수를 호출
function (value, index, array) { }
- map() : 콜백 함수에서 리턴한 값들을 기반으로 새로운 배열을 만드는 함수, 콜백 함수 내부에서 value * value를 하고 있으므로 모든 배열의 요소를 제곱한 새로운 배열을 만든다.
- filter() : 콜백 함수에서 리턴하는 값이 true인 것들만 모아서 새로운 배열을 만드는 함수
function(value, index, array) { }형태의 콜백 함수를 사용하는 것이 기본이지만, value만 활용하므로 value만 매개변수로 넣었습니다.
* 원하는 매개변수만 받기, forEach(), map() 함수의 완전한 형태를 보여드리고자 콜백 함수에 매개변수를 value, index, array로 3개를 모두 입력했지만, 일반적으로 value만 또는 value와 index만 사용하는 경우가 많다. 콜백 함수의 매개변수는

모두 입력할 필요 없고, 사용하고자 하는 위치의 것만 순서에 맞춰 입력하면 된다.
화살표(arrow) 함수 : map(), filter() 함수처럼 단순한 형태의 콜백 함수를 쉽게 입력하고자 생성 function 키워드 대신 화살표를 사용한다.
(매개변수) => {

}
(매개변수) => 리턴값
내부에서 this 키워드가 지칭하는 대상이 다르다는 등의 미세한 차이가 있다.
- 메소드 체이닝(method chaining) : 메소드가 리턴하는 값을 기반으로 해서 함수를 줄줄이 사용하는 것
타이머(timer) 함수 : 특정 시간마다 또는 특정 시간 이후에 콜백 함수를 호출할 수 있는 함수
setTimeout(함수, 시간) : 특정 시간 후에 함수를 한 번 호출합니다.
setInterval(함수, 시간) : 특정 시간마다 함수를 호출합니다.
clearTimeout(타이머_ID) : setTimeout() 함수로 설정한 타이머를 제거합니다.
clearInterval(타이머_ID) : setInterval() 함수로 설정한 타이머를 제거합니다.
* 즉시 호출 함수 : (function() {}) ()
스코프(scope) : 변수가 존재하는 범위, 스코프는 같은 단계에 있을 경우 무조건 충돌이 일어난다. 스코프 단계를 변경하는 방법은 중괄호를 사용해서 블록을 만들거나, 함수를 생성해서 블록을 만드는 방법
섀도잉(shadowing) : 블록이 다른 경우 내부 변수가 외부 변수를 가리는 현상을 조금 어려운 표현
- var 키워드 : 구 버전의 자바스크립트에서 변수를 선언할 때 사용하던 키워드는 함수 블록을 사용하는 경우에만 변수 충돌을 막을 수 있습니다.
- 트랜스파일러 : 구 버전의 자바스크립트를 지원하는 웹 브라우저에 대응해야 하는 경우가 많고, Babel 등 최신 버전의 자바스크립트를 구 버전의 자바스크립트로 변경해줌. 단순한 블록으로 함수 충돌을 막는 코드는 제대로 변환해주지 못한다.
* 엄격 모드 : 여러 자바스크립트 코드를 보면 블록의 가장 위쪽에 'use strict' 라는 문자열이 등장하는 것을 볼 수 있다. 자바스크립트는 문자열을 읽어들인 순간부터 코드를 엄격하게 검사한다.
<script>
'use strict'
문장
문장
</script>
* 익명 함수와 선언적 함수의 차이
while 반복문은 조건을 중심으로 반복할 때, for 반복문은 횟수를 중심으로 또는 배열 등을 중심으로 반복할 때 사용
- 익명 함수 : 순차적인 코드 실행에서 코드가 해당 줄을 읽을 때 생성
- 선언적 함수 : 순차적인 코드 실행이 일어나기 전에 생성, 같은 블록이라면 어디에서 함수를 호출해도 상관없다.
- 선언적 + 익명 함수 : 익명 함수는 우리가 코드를 읽을 때와 같은 순서로 함수가 선언되지만, 선언적 함수는 우리가 코드를 읽는 순서와 다른 순서로 함수가 선언된다. 함수를 같은 이름으로 덮어쓰는 것은 굉장히 위험한 일이다. 그래서 안전하게 사용할 수 있는 익명 함수를 더 선호
- 블록이 다른 경우에 선언적 함수 : 선언적 함수는 어떤 코드 블록(script 태그 또는 함수 등으로 구분되는 공간)을 읽어들일 때 먼저 생성, 다른 프로그래밍 언어들은 일반적으로 선언적 함수 형태로 함수를 많이 사용하지만, 자바스크립트는 블록이 예상하지 못하게 나뉘는 문제 등이 발생할 수 있어 안전을 위해 익명 함수를 더 많이 사용

Chapter 06. 객체

06-1. 객체의 기본

객체(object) : 추상적 의미, 실제로 존재하는 사물을 의미하고 이름과 값으로 구성된 속성을 가진 자바스크립트의 기본 데이터 타입, 여러 자료를 다룰 때 사용

배열을 typeof로 실행해 보면 object

배열에는 인덱스와 요소가 있다. 각각의 요소를 사용하려면 다음처럼 배열 이름 뒤에 인덱스로 접근

- 객체는 중괄호 {…}로 생성 키: 값

product['제품명'] / product.제품명 모두 객체의 요소에 접근할 수 있다.

* 식별자로 사용할 수 없는 단어를 키로 사용할 경우 : 객체를 생성할 때 키는 식별자와 문자열을 모두 사용할 수 있다. 대부분의 개별자가 식별자를 키로 사용하지만, 식별자로 사용할 수 없는 단어를 키로 사용할 때는 문자열을 사용해야 한다. 그리고 식별자가 아닌 문자열을 키로 사용했을 때는 무조건 대괄호[…]를 사용해야 객체의 요소에 접근할 수 있다.

속성과 메소드

- 요소 : 배열 내부에 있는 값
- 속성 : 객체 내부에 있는 값
- 메소드 : 객체의 속성 중 함수 자료형인 속성

요소와 속성을 표현하는 방법은 다르지만, 내부적으로 의미는 큰 차이가 없다.

- 메소드 내부에서 this 키워드 사용 : 메소드 내에서 자기 자신이 가진 속성을 출력하고 싶을 때는 자신이 가진 속성임을 분명하게 표시해야 합니다. 자기 자신이 가진 속성이라는 것을 표시

동적으로 객체 속성 추가 : 객체를 생성한 후 속성을 지정하고 값을 입력

동적으로 객체 속성 제거 : 객체의 속성을 제거할 때는 delete 키워드를 사용

delete 객체.속성

06-2. 객체의 속성과 메소드 사용하기

객체 자료형

- 객체 : 속성과 메소드를 가질 수 있는 모든 것, 배열도, 함수도 객체다.
- typeof 연산자를 사용해서 배열의 자료형을 확인해보면 "object"라고 객체가 출력되지만, 배열인지 확인하려면 Array.isArray() 메소드를 사용한다. (Array도 메소드를 갖고 있으므로 객체다.)
- 함수는 '실행이 가능한 객체'라는 특이한 자료로 typeof 연산자를 사용해서 자료형을 확인하면 "funtion"을 출력한다.
- 일급 객체(first-class bojext) : 함수는 객체의 특성을 완벽하게 갖고 있어 자바스크립트에서 함수를 일급 객체에 속한다고 표현하기도 한다.

기본 자료형 : 실체가 있는 것(undefined와 null 등이 아닌 것) 중에 객체가 아닌 것, 숫자, 문자열, 불

이러한 자료형은 객체가 아니므로 속성을 가질 수 없다.

기본 자료형을 객체로 선언하기

- const 객체 = new 객체 자료형 이름()
- 숫자 -> Number ex) const f = new Number(10)
- 문자 -> String ex) const f = new String('안녕하세요')
- 불 -> Boolean ex) const f = new Boolean(true)

기본 자료형의 일시적 승급

- 자바스크립트는 사용의 편리성을 위해서 기본 자료형의 속성과 메소드를 호출할 때(기본 자료형 뒤 온점 찍고 무언가 하려고 하면) 일시적으로 기본 자료형을 객체로 승급시킨다. 그래서 속성과 메소드를 사용할 수 있다.

> const h = '안녕하세요'

undefined

> h.sample = 10

10

-> 일시적으로 객체로 승급되어 sample 속성을 추가할 수 있습니다.

> h.sample

undefined

-> 일시적으로 승급된 것이라 추가했던 sample 속성은 이미 사라졌다.

- 기본 자료형의 경우 속성과 메소드를 사용할 수는 있지만, 속성과 메소드를 추가로 가질 수는 없다고 생각하면 된다.

프로토타입으로 메소드 추가하기

- prototype : 객체 전용 옷(틀), 객체에 속성과 메소드를 추가하면 모든 객체(와 기본 자료형)에서 해당 속성과 메소드를 사용할 수 있다.

- 객체 자료형 이름.prototype.메소드 이름 = function () {

}

- 모든 숫자 자료형이 어떤 값을 공유할 필요는 없으므로, 일반적으로 프로토타입에 속성을 추가하지 않지만, 프로토타입에 메소드를 추가하면 다양하게 활용할 수 있다.

제곱 연산자(**) : n제곱할 수 있다.

this.valueOf()로 숫자 값을 꺼낸다. this**n을 해도 아무 문제 없이 계산되지만, 객체 내부에서 값을 꺼내 쓰는 것임을 명확하게 하기 위해 valueOf() 메소드를 사용하는 것이 일반적이다.

indexOf() : 자바스크립트에서 문자열 내부에 어떤 문자열이 있는지, 배열 내부에 어떤 자료가 있는지 확인, 만약 해당 문자열이 있는지 확인할 수 있다. 문자열이 있으면 시작하는 위치(인덱스)를 출력하고, 없으면 -1을 출력한다.

문자열.indexOf(문자열) >= 0 등의 코드를 사용하면 문자열 내부에 어떤 문자열이 포함되어 있는지 ture 또는 false로 얻을 수 있다. 문자열.contain(문자열)을 했을 때 true 또는 false를 리턴하는 형태로 변경하면 편리하게 사용할 수 있다.

- Number 객체

숫자 N번째 자릿수까지 출력하기 : to Fixed()

소수점 아래 2자리까리 출력하고 싶다면 to Fixed(2), 3자리까지 출력하고 싶다면 to Fixed(3)

NaN과 Infinity 확인하기 : isNaN(), isFinite()

어떤 숫자가 NaN(Not a Number)인지 확인 할 때는 Number.isNaN()

NaN과 비교하면 모든 값이 false로 나오므로 isNaN() 메소드를 사용해야 NaN인지 확인할 수 있다.

어떤 숫자가 무한인지 확인할 때는 Number.isFinite()

Infinity는 숫자를 0으로 나누는 것과 같이 무한대 숫자를 의미, isFinite() 메소드가 false인 경우는 양의 무한대 숫자와 음의 무한대 숫자 두 가지 경우

이 메소드들은 숫자 자료 뒤에 온점을 찍고 사용하는 것이 아니라 Number 뒤에 점을 찍고 사용한다.

String 객체

- 문자열 양쪽 끝의 공백 없애기 : trim()

trim : 사용자의 실수 또는 악의적인 목적으로 문자열 앞뒤에 공백이 추가되는 경우가 많으므로 이런 것들을 미리 제거하는 것, trim() 메소드를 사용하면 문자열 앞뒤 공백(띄어쓰기, 줄 바꿈 등)을 제거할 수 있다.

“ 안녕하세요 ” -> “안녕하세요”

- 문자열을 특정 기호로 자르기 : split()

메소드는 문자열을 매개변수(다른 문자열)로 잘라서 배열을 만들어 리턴하는 메소드

JSON 객체 : 인터넷에서 문자열로 데이터를 주고 받을 때는 CSV, XML, CSON 등의 다양한 자료 표현 방식을 사용할 수 있다. 현재 가방 많이 사용되는 자료 표현 방식이다. JavaScript Object Notation의 약자로 자바스크립트의 객체처럼 자료를 표현하는 방식

- 값을 표현할 때는 문자열, 숫자, 불 자료형만 사용할 수 있습니다(함수 등은 사용 불가).

- 문자열은 반드시 큰따옴표로 만들어야 합니다.
- 키에도 반드시 따옴표를 붙어야 합니다.
- JSON.stringify() : 자바스크립트 객체를 JSON 문자열로 변환할 때 사용
- JSON.parse() : JSON 문자열을 자바스크립트 객체로 전개할 때 사용

Math 객체 : 수학과 관련된 기본적인 연산을 할 때, 객체 속성으로는 pi, e와 같은 수학 상수가 있다.
- 메소드 : Math.sin(), Math.cos(), Math.tan() 삼각함수
- Math.random() : 랜덤한 숫자를 생성할 때 사용, 메소드는 0 이상 , 1 미만의 랜덤한 숫자를 생성

0 <= 결과 < 1

외부 script 파일 읽어들이기 : 별도의 자바스크립트 파일을 만들어야 한다. html과 js 파일을 생성해서 같은 폴더에 넣어준다. 외부 자바스크립트 파일을 읽어들일 때도 script 태그를 사용하고, src 속성에 읽어들일 파일의 경로를 입력하면 된다.

Lodash 라이브러리
- 외부 라이브러리 : 다른 사람이 만든 다양한 함수와 클래스를 묶어서 제공해주는 것
- 유틸리티 라이브러리 : 개발할 때 보조적으로 사용하는 함수들을 제공해주는 라이브러리, underscor, Lodash 등

underscore는 _기호를 의미, Lodash는 Low Dash를 줄인 것이며, 마찬가지로 _기호를 의미하는 말
- [Full Build]를 클릭하면 웹 브라우저에 따라 발생하는 상황

곧바로 파일 다운로드가 될 경우 : 다운로드한 파일을 HTML 파일과 같은 위치에 놓고 읽어들인다.

파일 내용이 출력될 경우 : 마우스 오른쪽 버튼을 클릭하고 [다른 이름으로 저장]을 선택한 뒤 HTML 파일과 같은 위치에 놓고 다운로드 한다.
- CDN이란, 콘텐츠 전송 네트워크라는 의미, Lodash 라이브러리를 CDN 링크를 사용한다는 것은 이러한 곳으로부터 Lodash 파일을 읽어들여서 사용한다는 것
- min 버전 : 자바스크립트의 파일은 자바스크립트 코드를 집핑한 파일이다. 집핑이란, 코드를 응축하는 것
- sortBy() : 배열을 어떤 것으로 정렬할지 지정하면, 지정한 것을 기반으로 배열을 정렬해서 리턴해주는 메소드
- Luxon과 date-fns : 날짜와 시간을 쉽게 다루는 라이브러리
- Handsontable : 웹 페이지에 스프레드 시트를 출력하는 라이브러리
- D3.js와 ChartJS : 그래프를 그릴 수 있는 라이브러리
- Three.js : 3차원 그래픽을 다루는 라이브러리

06-3. 객체와 배열 고급

속성 존재 여부 확인
- 객체에 없는 속성에 접근하면 undefined 자료형이 나온다.

객체의 특정 속성이 false로 변환될 수 있는 값(0, false, 빈 문자열 등)이 아닐 때와 같은 전제가 있어야 안전하게 사용할 수 있는 코드

```
// 객체 내부에 속성이 있는지 확인합니다.
if (object.name) {
  console.log( 'name 속성이 있습니다.' )
}  else {
  console.log( 'name 속성이 없습니다.' )
}

if (object.author) {
  console.log( 'author 속성이 있습니다.' )
}  else {
  console.log( 'author 속성이 없습니다.' )
```

}

// 객체 내부에 속성이 있는지 확인합니다.
object.name || console.log('name 속성이 없습니다.')
object.author || console.log('author 속성이 없습니다.')
false로 변환될 수 있는 값이 들어오지 않을 것이라는 전제가 있으면 구현, 실제로 개발자들이 이런 코드를 많이 활용!
// 객체의 기본 속성을 지정합니다.
object.name = object.name || '제목 미정'
object.author = object.author || '저자 미상'

배열 기반의 다중 할당

- 다중 할당 : [식별자, 식별자, 식별자, ...] = 배열

할당 연산자(=) 왼쪽에 식별자(변수 또는 상수)의 배열을 넣고, 오른쪽에 배열을 위치시키면 배열의 위치에 맞게 값들이 할당됩니다. 배열의 크기는 같을 필요도 없고, const 키워드로도 사용할 수 있다.

객체 기반의 다중 할당

- 객체 속성 꺼내서 다중 할당하기 :

{속성 이름, 속성 이름} = 객체

{식별자 = 속성 이름, 식별자 = 속성 이름} = 객체

배열 전개 연산자

배열과 객체는 할당할 때 얕은 복사라는 것이 이뤄진다.

- 얕은 복사(참조 복사) : 복사하는 행위가 단순하게 다른 이름을 붙이는 형태로 동작하는 복사
- 깊은 복사 : 얕은 복사의 반대말, 복사한 두 배열이 완전히 독립적으로 작동, 클론(clone)을 만드는 것이라고 표현하기도 한다.
- 전개 연산자를 사용한 배열 복사 : [...배열]

두 배열이 독립적으로 작동

- 전개 연산자를 사용한 배열 요소 추가 : [...배열, 자료, 자료, 자료]

객체 전개 연산자

- 전개 연산자를 사용한 객체 복사 : {...객체}
- 전개 연산자를 사용한 객체 요소 추가 : {...객체, 자료, 자료, 자료}

Chapter 07. 문서 객체 모델

07-1. 문서 객체 조작하기

문서 객체(document object) : HTML 페이지에 있는 html, head, body, title, h1, div, span 등을 HTML 언어에서는 '요소'라고 한다. 그리고 자바스크립트에서는 이러한 요소들을 문서 객체하고 한다. 즉, 문서 객체를 조작한다는 것은 HTML 요소들을 조작한다는 의미이다.

문서 객체 모델(DOM, document objects model) : 문서 객체를 조합해서 만든 전체적인 형태

DOMContentLoaded 이벤트 : 웹 브라우저가 문서 객체를 모두 읽고 나서 실행하는 이벤트

! 코드 입력 시, DOMContentLoaded 문자열은 오탈자를 입력해도 오류를 발생하지 않습니다.

```
document.addEventListener('DOMContentLoaded', () => {
        // 문장
})
```

참고로, HTML 페이지는 코드를 위에서 아래로 차례대로 실행

* addEventListener() 메소드 : document.addEventListener('DOMContentLoaded', () => {})는 document라는 문서 객체의 DOMContentLoaded 이벤트가 발생했을 때, 매개변수로 지정한 콜백 함수를 실행해라는 의미입니다.

* load 이벤트 : DOMContentLoaded는 HTML5부터 추가된 이벤트로, HTML로 DOM 트리를 만드는 게 완성되었을 뿐만 아니라 이미지, 스타일시트 같은 외부 자원도 모두 불러오는 것이 끝났을 때 발생합니다.

문서 객체 가져오기

- document.querySelector(선택자) : 요소를 하나만 추출
- document.querySelectorAll(선택자) : 문서 객체를 여러 개 추출, 문서 객체 여러 개를 배열로 읽어 들여 내부의 요소에 접근하고 활용하려면 반복을 돌려야 한다. 일반적으로 forEach() 메소드를 사용해서 반복을 돌린다.
- 선택자 부분에는 css 선택자를 입력한다.

이름	선택자 형태	설명
태그 선택자	태그	특정 태그를 가진 요소를 추출
아이디 선택자	#아이디	특정 id 속성을 가진 요소를 추출
클래스 선택자	.클래스	특정 class 속성을 가진 요소를 추출
속성 선택자	[속성=값]	특정 속성 값을 갖고 있는 요소를 추출
후손 선택자	선택자_A 선택자_B	선택자_A 아래에 있는 선택자_B를 선택

글자 조작하기

- 문서 객체.textContent : 입력된 문자열을 그대로 넣습니다.
- 문서 객체.innerHTML : 입력된 문자열을 HTML 형식으로 넣습니다.

* innerText와 textContent : textContent 속성은 최신 웹 브라우저 또는 익스플로러 9 이후의 웹 브라우저에서 사용할 수 있는 속성이며, innerText는 속성의 성능 문제로 9 이전의 익스플로러에서 사용했다. 글자를 조작할 때는 성능이 좋은 textContent 속성을 사용하는 것이 좋다.

속성 조작하기

- 문서 객체.setAttribute(속성 이름, 값) : 특정 속성에 값을 지정, 리턴 타입은 void
- 문서 객체.getAttribute(속성 이름) : 특정 속성을 추출, 리턴 타입은 Object

setAttribute() 메소드의 value 파라미터의 타입이 Object이고, getAttribute() 메소드의 리턴 타입이 Object인데, 이것은 모든 클래스 타입을 속성의 값으로 사용 가능하다는 것을 의미한다.

스타일 조작하기 : 문서 객체의 스타일을 조작할 때는 style 속성을 사용한다.

- 캐멀 케이스(CamelCase) : 단어의 첫 글자를 대문자로 쓰는 것

CSS 속성 이름	자바스크립트 style 속성 이름
background-color	backgroundColor
text-align	textAlign
font-size	fontSize

자바스크립트에서는 - 기호를 식별자에 사용할 수 없어, 두 단어 이상의 속성은 캐멀 케이스로 나타낸다.

- h1.style.backgroundColor
- h1.style['backgroundColor']
- h1.style['background-color']

문서 객체 생성하기

- document.createElement(문서 객체 이름)
- 문서를 어떤 문서 아래에 추가할지를 지정해줘야 한다. 이러한 그림을 프로그래밍에서는 '트리'라고 한다. 어떤 문서 객체가 있을 때 위에 있는 것을 '부모', 아래 있는 것을 '자식'이라고 한다.
- 문서 객체 트리 구조 : 부모 객체.appendChild(자식 객체)

문서 객체 이동하기

- appendChild() 메소드 : 문서 객체를 이동할 때도 사용 가능하다. 문서 객체의 부모는 언제나 하나여야 하고, 문서 객체를 다른 문서 객체에 추가하면 문서 객체가 이동한다.

문서 객체 제거하기 : removeChile() 메소드를 사용

- 부모 객체.removeChile(자식 객체)
- 문서 객체.parentNode.removeChild(문서 객체)

이벤트 설정하기

- 모든 문서 객체는 생성되거나 클릭되거나 마우스를 위에 올리거나 할 때 이벤트(event)가 발생하고, 이벤트가 발생할 때 실행할 함수는 addEventListener() 메소드를 사용
- 문서 객체.addEventListener(이벤트 이름, 콜백 함수)
- 이벤트 리스너(이벤트 핸들러) : 이벤트가 발생할 때 실행할 함수 -> 콜백 함수
- 이벤트를 제거할 때 사용하는 함수는 removeEventListener() 메소드를 사용
- 문서 객체.removeEventListener(이벤트 이름, 이벤트 리스너)

07-2. 이벤트 활용

이벤트 모델 : 이벤트를 연결하는 방법

- 이벤트를 연결할 때 사용하는 메소드 addEventListent() : 표준 이벤트 모델
- 고전 이벤트 모델 : on○○으로 시작하는 속성에 함수를 할당해서 이벤트를 연결하는 방법
- 인라인 이벤트 모델 : on○○으로 시작하는 속성을 HTML 요소에 직접 넣어서 이벤트를 연결하는 것
- 이벤트 객체 : 이벤트 리스너의 첫 번째 매개변수로 이벤트와 관련된 정보가 들어있다.

키보드 이벤트

이벤트	설명
keydown	키가 눌릴 때 실행. 키보드를 꾹 누르고 있을 때도, 입력될 때도 실행
keypress	키가 입력되었을 때 실행되지만, 웹 브라우저에 따라서 아시아권의 문자를 제대로 처리하지 못하는 문제가 있다.
keyup	키보드에서 키가 떨어질 때 실행

keydown 이벤트와 keypress 이벤트는 웹 브라우저에 따라 아시아권의 문자(한국어, 중국어, 일본어)를 제대로 처리하지 못하는 문제가 있어 일반적으로 keyup 이벤트를 사용한다.

- 키보드 키 코드 사용하기

이벤트 속성 이름	설명
code	입력한 키
keyCode	입력한 키를 나타내는 숫자
altKey	Alt 키를 눌렀는지
ctrlKey	Ctrl 키를 눌렀는지
shiftKey	Shift 키를 눌렀는지

code 속성은 입력한 키를 나타내는 문자열이 들어있고, altKey, ctrlKey, shiftKey 속성은 해당 키를 눌렀는지 불 자료형 값이 들어있다.

- code 속성값 : https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/code/code_values
- keyCode 속성값 : https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent/keyCode

이벤트 발생 객체

- event.currentTarget 속성을 사용한다. 이는 () => {}와 function () {} 형태 모두 사용 가능
- this 키워드를 사용한다. 화살표 함수가 아닌 function () {} 형태로 함수 선언한 경우에 사용

글자 입력 양식 이벤트

- 입력 양식 : 사용자로부터 어떠한 입력을 받을 때 사용하는 요소, HTML에서는 input 태그, textarea 태그, button 태그, select 태그 등이 모두 입력 양식이다.
- change 이벤트 : 원래 입력 양식은 값이 변경될 때 change 이벤트를 발생합니다. 입력 양식을 선택(focus 상태)해서 글자를 입력하고 선택을 해제(blur 상태)할 때 change 이벤트를 발생합니다. 단, 사용자가 입력하는 중에는 change 이벤트가 발생하지 않습니다.
- 드롭다운 목록 활용하기 : 드롭다운 목록은 기본적으로 select 태그로 구현, select 태그에 multiple 속성을 부여하면 Ctrl 키 또는 Shift 키를 누르고 여러 항목을 선택할 수 있다.
- toFixed(?) 메소드 : 소수 ? 자리까지 출력
- 체크 박스 활용하기 : 체크 박스처럼 체크 상태를 확인할 때는 입력 양식의 checked 속성을 사용, change 이벤트가 발생했을 때 체크 박스의 체크 상태를 확인하고 setInterval() 함수 또는 clearInterval() 함수를 실행한다.
- 라디오 버튼 활용하기 : 체크 박스와 비슷한 입력 양식 요소이고, 체크 박스와 마찬가지로 checked 속성을 사용

! name 속성이 없는 라디오 버튼은 카테고리 구분 없이 선택할 수 있으며, 한 번 선택하고 나면 선택을 취소할 수도 없다. 따라서 라디오 버튼을 사용할 때는 꼭 name 속성과 함께 사용하기

기본 이벤트 막기

- 컨텍스트 메뉴 : 웹 브라우저 속 이미지에서 마우스 오른쪽 버튼을 클릭하면 나타나는 메뉴
- 기본 이벤트 : 어떤 이벤트가 발생했을 때 웹 브라우저가 기본적으로 처리해주는 것으로, 기본 이벤트를 제거할 때는 event 객체의 preventDefault() 메소드를 사용한다.

* localStorage 객체 : 웹 브라우저에 데이터를 저장하는 객체로 웹 브라우저가 기본적으로 제공하는 객체이다.

- localStorage.getitem(키) : 저장된 값을 추출합니다. 없으면 undefined가 나온다. 객체의 속성을 추출하는 일반적인 형태로 localStorage.키 또는 localStorage[키] 형태로 사용할 수도 있다.
- localStorage.setitem(키, 값) : 값을 저장한다. 객체에 속성을 지정하는 일반적인 형태를 사용할 수도 있다.
- localStorage.removeitem(키) : 특정 키의 값을 제거한다.
- localStorage.clear() : 저장된 모든 값을 제거한다.

* WebAPI : 웹 브라우저가 제공해주는 기능, https://developer.mozilla.org/ko/docs/Web/API

Chapter 08. 예외 처리

08-1. 구문 오류와 예외

구문 오류(syntax error) : 오류로 인해 코드가 실행조차 되지 않는 오류

예외 처리(exception handling) : 문법적 오류를 제외하고 코드 실행 중간에 발생하는 오류를 처리하는 것

오류의 종류

- 프로그램 실행 전에 발생하는 오류(구문 오류), 프로그램 실행 중에 발생하는 오류(예외, 런타임 오류)
- 구문 오류 : 괄호의 짝을 맞추지 않았다든지, 문자열을 열었는데 닫지 않았다든지 할 때 발생하는 오류, Syntax Error 오류가 발생한다.
- 예외 : 예외 또는 런타임 오류는 실행 중에 발생하는 오류로, Syntax Error 오류 이외의 모든 오류(TypeError, ReferenceError, RangeError)가 예외로 분류된다.

기본 예외 처리 : 조건문을 사용해서 예외가 발생하지 않게 만드는 것

! 자바스크립트는 다른 프로그래밍 언어와 비교해 굉장히 유연하기 때문에 예외를 발생할 가능성이 적은 편이지만, 프로그램에 문제가 발생했는데도 죽지 않고 실행되면 계속해서 만들 가능성이 있다. 따라서 문제가 발생할 수 있는 부분은 조건문 등으로 처리해줘야 한다.

고급 예외 처리

- try catch finally : 예외를 조금 더 쉽게 잡을 수 있는 기능, try 구문 안에서 예외를 발생하면 catch에서 처리, finally 구문은 필수 사항은 아니며 예외 발생 여부와 상관없이 수행해야 하는 작업이 있을 때 사용.

! 기본 예외 처리와 고급 예외 처리 중 어떤 것이 좋다고 딱단할 수는 없지만, 일반적으로 기본 예외 처리를 사용하는 것이 약간은 더 빠르지만, 컴퓨터의 성능이 너무 좋아져서 차이가 거의 없다. 편하다고 생각하는 방법을 사용하면 되고, 어떤 예외가 발생할지 예측하기 힘든 경우가 있다면 고급 예외 처리로 처리해주는 것이 좋다.

- try catch 구문 내부에서 return 키워드를 만나거나, break 또는 continue 키워드를 만날 때 결과가 달라진다.

Nodejs처럼 서버로 사용하는 자바스크립트에서는 이러한 내용을 알아야 안전하게 코드를 작성할 수 있다.

08-2. 예외 처리 고급

예외 객체(exception object) : 예외가 발생하면 예외와 발생된 정보를 확인할 수 있게 해주는 것, try catch 구문을 사용할 때 catch의 괄호 안에 입력하는 식별자를 말하기도 한다. 아무 식별자나 입력해도 괜찮지만, 일반적으로 e나 exception이라는 식별자를 사용한다.

```
try {
} catch (exception) {
}
```

- 예외 객체의 속성

속성 이름	설명
name	예외 이름
message	예외 메시지

예외 강제 발생 : throw 키워드 사용한다.

```
// 단순하게 예외를 발생시킵니다.
throw 문자열

// 조금 더 자세하게 예외를 발생시킵니다.
throw new Error(문자열)
```

- 코드 실행 중에 throw 키워드를 사용하면 예외를 발생하므로 프로그램이 중단된다. 개발할 때는 어떤 사람이 Lodash 라이브러리처럼 다양한 기능을 가진 유틸리티 함수(또는 클래스)를 만들고, 다른 사람들이 그러한 라이브러리의 함수(또는 클래스)를 활용하는 경우가 많기에, 내가 만든 함수를 다른 사람이 사용할 때는 내가 의도하지 않은 형태로 코드를 사용할 수

있기에 예외를 강제로 발생시키면 사용자에게 주의를 줄 수 있으며, 의도한 대로 처리하게 유도할 수 있다.

- object.a가 undefined로 나오며, object.b도 undefined로 나온다. 여기에서 undefined + undefined를 하면 NaN이 나온다. 즉, 아무 오류 없이 정상적으로 실행된다.

Chapter 09. 클래스

09-1. 클래스의 기본 기능

객체 지향(Object Oriented) 패러다임 : 객체를 우선적으로 생각해서 프로그램을 만든다는 방법론

객체 지향 프로그래밍 언어들은 클래스라는 문법으로 객체를 효율적이고 안전하게 만들어 객체 지향 패러다임을 쉽게 프로그래밍에 적용할 수 있도록 도와줍니다.

추상화(abstraction) : 프로그램에 필요한 요소만 사용해서 객체를 표현하는 것, 복잡한 자료, 모듈, 시스템 등으로부터 핵심적인 개념과 기능을 간추려내는 것

- 예를 들어, 성적 관리 프로그램을 만들 때, 학생이라는 객체가 필요하고 학생들로부터 성적 관리에 필요한 공통사항을 추출하는데 이를 추상화라고 한다. 학생들이 여러 명이므로 추출한 요소는 배열을 이용해 관리한다.

객체를 처리하는 함수

- 객체를 만드는 부분과 객체를 활용하는 부분으로 나누고, 이렇게 코드를 분할하면, 객체에 더 많은 기능을 추가하게 되었을 때 객체를 쉽게 유지보수할 수 있으며, 객체를 활용할 때도 더 간단하게 코드를 작성할 수 있습니다.

객체의 기능을 메소드로 추가하기

- 객체의 키와 값을 하나하나 모두 입력해서 생성하면 함수 이름 충돌도 발생하지 않고, 함수를 잘못 사용하는 경우도 줄일 수 있다.
- 함수를 사용해 객체를 찍어내면 객체를 좀 더 손쉽게 생성할 수 있다. 함수를 만들고 여기에 객체를 만들어 리턴해서 함수를 만들면 객체를 하나하나 만들 때와 비교해 오탈자의 위험이 줄어들고, 코드를 입력하는 양이 크게 줄어들고, 속성과 메소드를 한 함수 내부에서 관리할 수 있으므로 객체를 더 손쉽게 유지보수 할 수 있습니다.

클래스 선언하기

- 객체 지향 프로그래밍 : 객체들을 정의하고 그러한 객체를 활용해서 프로그램을 만드는 것

언어 : C++, 자바, 루비, 코틀린, PHP 등

- 클래스 : 객체를 만들 때 수많은 지원을 하는 대신 많은 제한을 거는 문법

```
class 클래스 이름 {
}
```

클래스 기반으로 만든 객체 = 인스턴스(객체)

```
new 클래스 이름()
```

* 클래스 이름은 첫 글자를 대문자로 지정하는 것이 개발자들의 약속이다. 첫 번째 글자를 소문자로 지정해도 오류를 발생하지 않지만, 식별자만 보고도 클래스라는 것을 바로 이해할 수 있도록 첫 글자를 대문자로 만드는 약속을 지키는 것이 좋다.

- 프로토타입 : 제한을 많이 하지만, 대신 지원도 별로 하지 않는 문법

생성자

```
class 클래스 이름 {
  constructor () {
    /* 생성자 코드 */
  }
}
```

- 생성자는 클래스를 기반으로 인스턴스를 생성할 때 처음 호출되는 메소드입니다. 따라서 생성자에서는 속성을 추가하는 등 객체의 초기화 처리를 한다.

메소드

- 메소드 사이에 쉼표를 넣으면 안된다.

09-2. 클래스의 고급 기능

상속 : 어떤 클래스가 갖고 있는 유산(속성과 메소드)을 다른 클래스에 물려주는 형태로, 클래스의 선언 코드를 중복해서 작성하지 않도록 함으로써 코드의 생산 효율을 올리는 문법이다.

class 클래스 이름 extends 부모클래스 이름 {
}
- 부모 클래스 : 유산을 주는 클래스
- 자식 클래스 : 유산을 받는 클래스
- 프로그램을 개발할 때 사용하는 거대한 규모의 클래스, 함수, 도구 등의 집합을 의미하는 프레임워크와 엔진이라는 것을 만드는 개발자와 이를 활용해서 다수를 대상으로 하는 서비스, 애플리케이션, 게임을 개발하는 개발자가 다른 경우가 많다. 전자를 프레임워크 개발자 또는 엔진 개발자라 하고, 후자를 애플리케이션 개발자 등으로 부릅니다.

private 속성과 메소드
- 클래스 사용자가 클래스 속성(또는 메소드)을 의도하지 않은 방향으로 사용하는 것을 막아 클래스의 안정성을 확보하기 위해 나온 문법

```
class 클래스 이름 {
  #속성 이름
  #메소드 이름 () {
  }
}
```

- 속성과 메소드 이름 앞에 #을 붙이기만 하면 된다. #이 붙어있는 속성과 메소드는 모두 private 속성과 메소드가 된다.

! private 속성은 사용하기 전에 미리 외부에 어떤 속성을 private 속성으로 사용하겠다고 선언해줘야 한다. private 속성을 사용하면 외부에서는 #length 속성에 아예 접근할 수 없는 문제가 발생한다.

게터와 세터
- 게터(getter) : get○○() 메소드처럼 속성 값을 확인할 때 사용하는 메소드
- 세터(setter) : set○○() 메소드처럼 속성에 값을 지정할 때 사용하는 메소드
- 게터와 세터는 필요한 경우에만 사용한다. 만약 사용자가 값을 읽는 것을 거부하겠다면 게터를 만들지 않아도 된다. 또한 사용자가 값을 지정하는 것을 거부하겠다면 세터를 만들지 않아도 된다.

```
class 클래스 이름 {
  get 이름 () { return 값 }
  set 이름 (value) { }
}
```

static 속성과 메소드
- 디자인 패턴 : 효율적으로 프레임워크를 개발할 수 있게 다양한 패턴을 고안하는데, 이러한 패턴을 말한다.

```
class 클래스 이름 {
  static 속성 = 값
  static 메소드() {

  }
}
```

static 속성과 메소드는 인스턴스를 만들지 않고 사용할 수 있는 속성과 메소드입니다. 변수와 함수처럼 사용할 수 있습니다.

클래스 이름.속성
클래스 이름.메소드()

* 오버라이드: 부모가 갖고 있는 함수를 자식에서 다시 선언해서 덮어쓰는 것으로 프레임워크를 다룰 때 반드시 활용하는 개념이다.

- toString() : 자바스크립트의 모든 객체는 이 메소드를 갖고 있다. 숫자, 문자열, 불, 배열, 함수, 클래스, 클래스의 인스턴스 모두 toString()이라는 메소드가 있습니다.

Chapter 10. 리액트 라이브러리 맛보기

10-1. 리액트의 기본

리액트 라이브러리(React Livrary) : 규모가 큰 자바스크립트 라이브러리로, 사용자 인터페이스(UI)를 쉽게 구성할 수 있도록 도와줍니다.

https://ko.reactjs.org/

리액트 라이브러리 사용 준비하기 : HTML 파일에서 다음과 같은 3개의 자바스크립트를 읽어들이는 것입니다.

https://unpkg.com/react@17.0.2/umd/react.development.js

https://unpkg.com/react-dom@17.0.2/umd/react-dom.development.js

https://unpkg.com/babel-standalone@6.26.0/babel.min.js

- 바벨(Babel) : 자바스크립트가 아니라 리액트를 위해서 개발된 자바스크립트 확장 문법, 라이브러리를 추가로 읽어들이고 바벨을 적용할 부분을 지정해야 한다.

루트 컴포넌트 출력하기

- 리액트(React) : 사용자 인터페이스를 쉽게 구성할 수 있게 도와주는 라이브러리
- 컴포넌트(component) : 리액트에서는 화면에 출력되는 요소
- 루트 컴포넌트(root component) : 가장 최상위에 배치하는 컴포넌트
- 컴포넌트 생성하기

<컴포넌트 이름> </컴포넌트 이름>

이렇게 컴포넌트를 출력할 때는 ReactDOM.render() 메소드를 사용

- 컴포넌트 출력하기

ReactDOM.render(컴포넌트, 컨테이너)

컨테이너는 컴포넌트를 출력할 상자

- JSX(자바스크립트 확장 문법) : React에서 HTML을 표현할 때, JSX를 사용한다. 외관상 HTML같은 마크업 언어를 리터럴로 입력하는 것으로 보이는데, 빌드 시 Babel에 의해 자바스크립트로 변환된다. 바벨 REPL 도구를 사용하면 바벨이 어떤 식으로 코드를 바꾸는지 확인할 수 있습니다.

바벨 REPL : https://babeljs.io/repl

JSX 기본 문법 :단순하게 태그를 만드는 기능 이외에도 태그 내부에 표현식을 삽입해서 출력하는 기능도 제공한다.

- 표현식 출력하기

<태그>{표현식}</태그>

<태그 속성={표현식} />

표현식을 출력할 때는 따옴표를 사용하면 안된다.

클래스 컴포넌트 :클래스로 만든 컴포넌트

- 함수 컴포넌트 : 함수로 만드는 컴포넌트
- 클래스 컴포넌트 만들기

```
class 컴포넌트 이름 extends React.Component
  render () {
    return <h1> 출력할 것 </h1>
  }
}
```

React.Component 클래스의 상속을 받아야 동작할 수 있게 속성과 메소드를 받을 수 있습니다. 화면에 무언가를 출력할 때 render() 메소드를 호출합니다. 이를 오버라이드해서 원하는 것을 출력합니다.

컴포넌트의 기본적인 속성과 메소드

- React.Component 클래스는 여러 속성과 메소드를 제공해준다. 이러한 속성을 변경하고 메소드를 오버라이드하고 우리가 필요한 속성과 메소드를 클래스에 추가해서 컴포넌트를 만든다.

- 클래스의 메소드 오버라이드하기

```
class App extends React.Component {
  constructor (props) {
    super(props)
    // 생성자 코드
    }

  render () {
   // 출력할 것
  }

  componentDidMount () {
   // 컴포넌트가 화면에 출력될 때 호출
  }

  componentWillUnmount () {
   // 컴포넌트가 화면에서 제거될 때 호출
  }
}
```

우리가 변경해서 사용하는 속성으로는 state 속성, 속성 값을 변경할 때는 반드시 setState() 메소드를 사용한다. 속성의 값을 변경하면 컴포넌트는 render() 메소드를 호출해서 화면에 변경사항을 출력합니다.

```
// 상태 선언하기(생성자 위치)
this.state = { 속성: 값 }
// 상태 변경하기(이외의 위치)
this.setState({ 변경할 속성: 값 })
```

이벤트 연결하기

- 메소드를 선언하고, 메소드에 this를 바인드하고, render() 메소드에서 출력하는 태그의 이벤트 속성에 메소드를 입력해서 이벤트를 연결합니다.

```
class App extends React.Component {
  constructoer (props) {
    super(props)
    this.메소드 이름 = this.메소드 이름.bind(this)
  }

  render () {
    return <h1 이벤트 이름={this.메소드 이름}></h1>
  }

  메소드 이름 (event) {
   // 이벤트가 호출될 때 실행할 코드
  }
}
```

this.메소드 이름 = this.메소드 이름.bind(this)는 리액트에서 이벤트를 연결할 때 반드시 사용해야 하는 코드로, 사용하지 않으면 이벤트 핸들러에서 this를 입력했을 때 undefined(또는 비전에 따라서 window 객체)가 나온다.

* 리액트 이벤트 이름을 확인할 수 있는 주소 : 기본적으로 onMouseDown, onMouseEnter, onDragEnd처럼 캐멀 케이스를 사용하면 되지만, 가끔 어떻게 입력해야 하는지 헷갈릴 때가 있습니다.

https://ko.reactjs.org/docs/events.html#clipboard-events

스타일 지정하기

```
render () {
  const style = { }
  return <h1 style={style}>글자</h1>
}
```

문서 객체 모델에서 본 것과 마찬가지로 style 객체에는 캐멀 케이스로 속성을 입력합니다. 크기 등의 단위는 숫자만 입력하면 된다. 단위를 입력하지 않아도 된다.

CSS 스타일 속성 이름	가능한 형태(1)	가능한 형태(2)

color: red	{
	color: 'red'
	}
		{
		'color': 'red'
		}

font-size: 2px	{
	fontSize: 2
	}
		{
		'font-size':2
		}

컴포넌트 배열 : 컴포넌트를 요소로 갖는 배열을 사용해서 한 번에 여러 개의 컴포넌트를 출력할 수 있다.

10-2. 리액트와 데이터

Flux 패턴 : 단방향 데이터 흐름이다. 데이터 흐름은 항상 Dispatcher에서 Store로, Store에서 View로, View는 Action을 통해 다시 Dispatcher로 데이터가 흐르게 된다. 이런 단방향 데이터 흐름은 데이터 변화를 훨씬 예측하기 쉽게 만든다. Flux를 크게 Dispatcher, Store, View 세 부분으로 구성된다.

여러 개의 컴포넌트 사용하기 : 컴포넌트1에서 컴포넌트2로 value 속성을 전달하고, 컴포넌트2에서 value 속성을 출력할 수 있다.

부모에서 자식의 state 속성 변경하기 : 부모 컴포넌트에서 자식 컴포넌트로 어떤 데이터를 전달할 때는 속성(this.props)을 사용합니다. 부모 컴포넌트에서 자식으로 어떤 데이터를 전달한 뒤 화면 내용을 변경할 때도 속성(this.props)를 사용합니다.

- componentDidUpdate() 메소드 : 컴포넌트에 변경이 발생했을 때 호출되는 메소드, 오버라이드해서 사용하고 있다. 매개변수로 변경 이전의 속성(prevProps)가 들어온다. 이 속성 값과 현재 속성 값을 비교해서 변경이 있는 경우(다른 경우)에만 setState() 메소드를 호출해서 화면에 변경사항을 출력합니다. 메소드 부분이 없으면 시간은 변하지 않습니다.
- render() 메소드는 단순하게 컴포넌트를 조합해서 문서 객체를 만든 뒤 화면에 출력하는 메소드가 아니다. 내부적으로 쓸데없는 변경 등을 막아 애플리케이션의 성능을 높일 수 있게 다양한 처리를 해준다.

자식에서 부모의 state 속성 변경하기 : 자식 컴포넌트에서 부모 컴포넌트의 상태를 변경할 때는 메소드를 사용한다. 부모 컴포넌트에서 자신(부모)의 속성을 변경하는 메소드를 자식에게 전달한 뒤, 자식에서 이를 호출하게 만드는 것이다.

※ 한빛미디어 '혼자 공부하는 자바스크립트' 의 내용을 바탕으로 정리한 내용입니다.