

javascript



- 1 자바스크립트 기초
- 2 자료와 변수
- 3 조건문
- 4 반복문
- 5 함수

6 객체

7 DOM

8 예외처리

9 클래스

10 기타

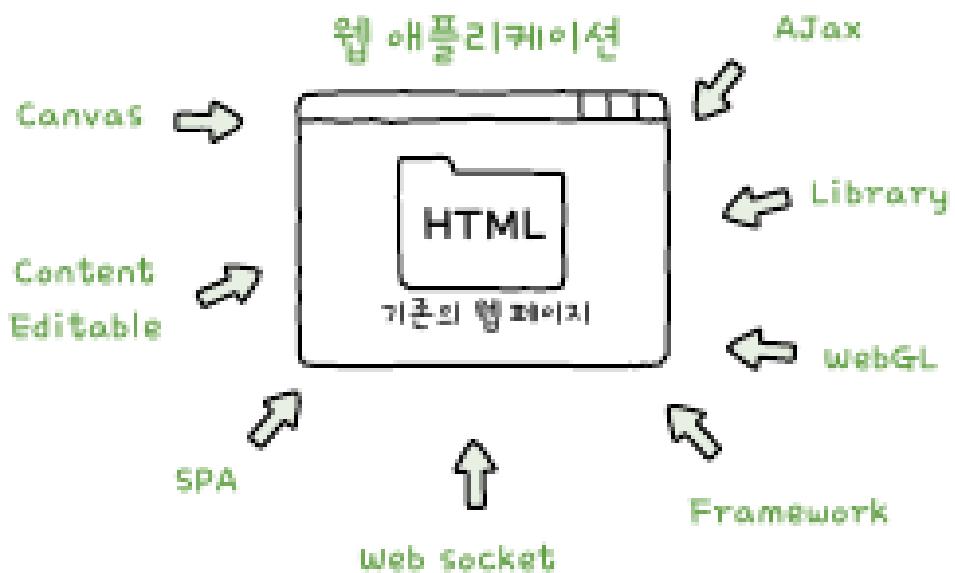
chapter 1 자바스크립트 기초

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. A white rectangular overlay box is positioned in the center-left area of the image. Inside this box, the text "javascript 활용" is written in a large, bold, black sans-serif font.

javascript 활용

- 웹 브라우저에서 사용하는 프로그래밍 언어.
- 브랜드 아이크에 의해서 제작, 처음 이름은 Mocha.
- 이후 livescript.
- 썬마이크로 시스템즈와 함께 javascript라는 이름으로 발전.
- 왜?
- 정적인 웹 문서의 내용을 동적으로 바꾸거나 이벤트 처리가 가능해짐.

- 단순 링크만을 의미 하지 않음.
- 다양한 기능을 가지게 되면서, 웹 애플리케이션으로 지칭.



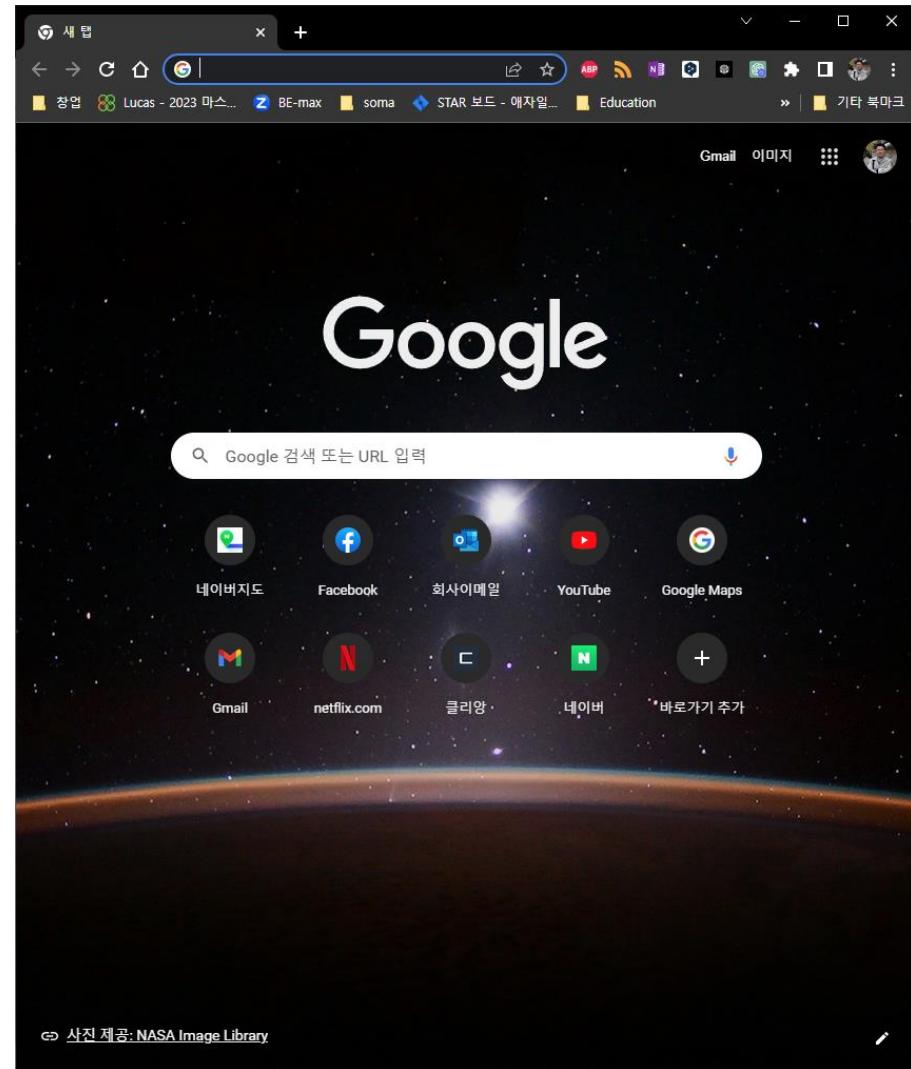
- 웹프론트와 백엔드는 이종의 언어로 개발
 - 백엔드 : java, c#, python, ruby....
 - 프론트엔드: javascript
- 2009년 node.js 등장으로 javascript로 FE/BE 다 개발이 가능해짐.
- 실제 linkedin에서 ruby로 개발된 서버를 node.js로 교체하고 서버 1/10로 줄임. 속도는 20배 빨라짐.
- full stack이라는 말이 나오는 게 가능해짐.
- 심지어 IOS/Android 개발도 가능한 reactive Native로 모바일 개발도 할 수 있음.
- Node webkit을 통해서 모바일 이외에 데스크탑 어플도 가능.

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. The background is blurred.

개발환경

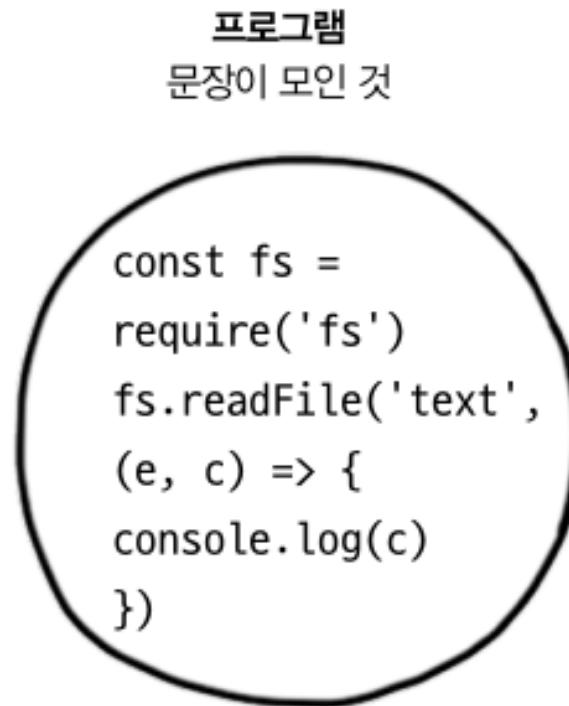
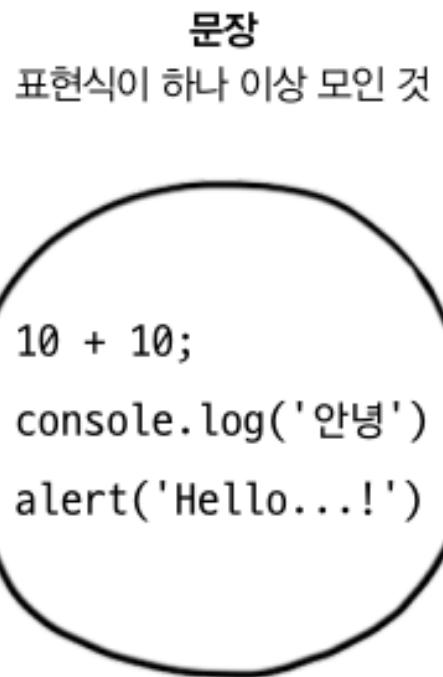
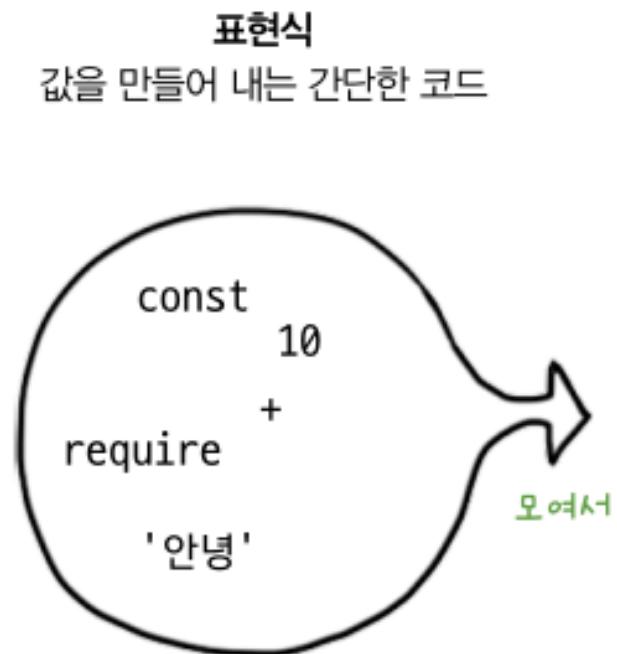
텍스트에디터와 코드 실행기

```
<!DOCTYPE html>
<html>
<head>
    <title>Media Query Basic</title>
    <link rel="stylesheet" href="desktop.css" media="screen" />
    <link rel="stylesheet" href="print.css" media="print" />
</head>
<body>
    <h1>Lorem ipsum</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis<br/>Aenean luctus congue scelerisque. Maecenas aliquet ante elemen</p>
</body>
</html>
```



A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. A large, semi-transparent white rectangular box covers the center of the image, containing the Korean text "용어".

용어



- 값을 만들어내는 간단한 코드 : expression 표현식
 - 273
 - 10+20
 - 'RintlanTta'
- 하나이상의 표현식이 모인것 : Statement 문장. 끝에는 ; (세미콜론) 또는 줄바꿈
 - 10+20+30;var rintiantta = 'Rint'+ 'lan' + 'Tta';
 - 10+20+30
 - let rintiantta = 'Rint'+ 'lan' + 'Tta'
- 줄바꿈과 세미콜론을 2가지 다 입력하는 경우도 있음.
- 이 문장을

- 처음 만들어질 때 정해놓은 특별한 의미가 있는 단어.

- await
- break
- case
- class
- const
- function
- instanceof

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

- 프로그래밍 언어에서 이름을 붙일 때 사용하는 단어.
- 변수명이나 함수명으로 사용.
 - 키워드를 사용하면 안됨.
 - 숫자로 시작하면 안됨.
 - 특수문자는 _ 와 \$만 허용
 - 공백문자를 포함할 수 없음.
- 일반적인 관례
 - 클래스 이름은 항상 대문자로 시작.
 - 변수와 인스턴스, 함수, 메소드의 이름은 항상 소문자로 시작
 - 여러 단어로 이루어진 경우 식별자는 각 단어의 첫 글자를 대문자로.

구분	단독으로 사용	다른 식별자와 사용
식별자 뒤에 괄호 없음	변수	속성
식별자 뒤에 괄호 있음	함수	메소드

<code>alert('Hello World')</code>	→ 함수
<code>Array.length</code>	→ 속성
<code>input</code>	→ 변수
<code>prompt('Message', 'Defstr')</code>	→ 함수
<code>Math.PI</code>	→ 속성
<code>Math.abs(-273)</code>	→ 메소드

chapter 2

자료와 변수

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. The background is blurred.

기본자료형

- 프로그램이 처리할 수 있는 모든 것들을 data.
- 자료형태에 따라 나눠 놓은 것을 data type.
- number, string, boolean

- 문자의 집합을 문자열, string 이라고 함.
- 문자가 하나라도 문자열 자료형.
- 만드는 방법
 - " "
 - ' '
- 그럼 따옴표를 출력하고자 할때는? 이스케이프 문자(\)를 사용.
 - \n : 줄바꿈
 - \t : 탭을 의미
 - \\ : 역슬래쉬 자체.

- 문자열 연결 연산자

- 문자열 사이 덧셈 기호 (+)를 사용하면 문자열을 연결할 수 있음.
- 문자열 연결 연산자라고 함.
- 문자열 + 문자열 → 문자열 연결 연산자.

- 문자열 선택 연산자

- 문자열[숫자]
- 문자 인덱스

안	녕	하	세	요
[0]	[1]	[2]	[3]	[4]

- 길이

- '문자열'.length (3)

- 숫자를 입력하면 숫자 자료형.
- 자바스크립트는 소수점이 있는 숫자와 없는 숫자를 모두 같은 자료형으로 인식.
- 숫자 연산자

연산자	설명	연산자	설명
+	더하기 연산자	*	곱하기 연산자
-	빼기 연산자	/	나누기 연산자

- 연산자 우선순위
 - 곱셈 먼저, 하고 더하기.
- 나머지 연산자
 - % - 좌변을 우변으로 나눈 나머지 출력

- 조건문 괄호안에서 많이 사용.
- 논리 부정 연산자 !

연산자	설명
<code>==</code>	양쪽이 같다.
<code>!=</code>	양쪽이 다르다.
<code>></code>	왼쪽이 더 크다.
<code><</code>	오른쪽이 더 크다.
<code>>=</code>	왼쪽이 더 크거나 같다.
<code><=</code>	오른쪽이 더 크거나 같다.

- `&&` : 양쪽값이 true 일때
만 true
- `||` : 양쪽값중 하나라도
true여도 true

연산자	설명
<code>&&</code>	논리곱 연산자
<code> </code>	논리합 연산자

&& 연산자

좌변	우변	결과
True	True	True
True	False	False
False	True	False
False	False	False

|| 연산자

좌변	우변	결과
True	True	True
True	False	True
False	True	True
False	False	False

- typeof(자료)
 - typeof('문자열') string
 - typeof(273) number
 - typeof(true) boolean
 - typeof '문자열' string
- 자료형
 - string, number, boolean, undefined, function, object, symbol, bigint

- 백틱 (`) 기호를 이용해서 감싸주고, \${ } 로 표현식을 만들어줌.
- `\${a+B}`
- 상황에 따라서 문자열을 치환하여야 하는 경우에는 템플릿을 이용해서 변환을 많이 함.

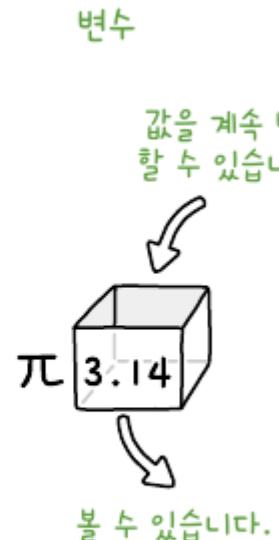
==연산자와 !=연산자

- ==연산자와 !=연산자는 "값과 자료형이 같은지" 비교
- ==연산자와 !=연산자는 "값이 같은지" 비교

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. The background is blurred.

상수와 변수

- 상수는 불변.
- 변수는 변할 수 있는 수



- 상수 constant.
- const 이름 = 값
- 사례
 - >const pi = 3.141492
 - >pi
 - 3.141592
 - >const r = 10
 - >2*pi*r
 - 62.83184

- Identifier has already declared
 - 상수는 한파일에서 한번만 선언 가능.
 - 한번 더 선언하면 오류발생.
- Missing initializer in const declaration
 - 상수는 반드시 선언할 때, 값을 지정해줘야 함.
 - 지정값을 안주면, 오류 발생. 변수와 다름.
- assignment to constant variable
 - 상수값을 변경하려고 시도하면 발생.
 - 상수값은 변경이 불가.
 - 변경할 값은 변수로 선언

- 변수 variable.
- let 이름 = 값
- 사례
 - >let pi = 3.141492
 - >pi
 - 3.141592
 - >let r = 10
 - >2*pi*r
 - 62.83184

- Identifier has already declared
 - 변수는 한파일에서 한번만 선언 가능.
 - 한번 더 선언하면 오류발생.
- @ 변경할 가능성이 있으면, 변수를 사용, 그렇지 않다면 상수를 사용.
- @ var : 예전에 사용. 변수를 중복해서 선언할 수 있다는 위험성, 변수가 속하는 범위가 애매하다는 이유로 let 키워드 등장으로 대체됨.

- 대입 연산자와 다른 연산자를 함께 사용하는 연산자.

복합 대입연산자	설명	사용예	의미
<code>+=</code>	기존변수의 값에 값을 더함.	<code>a+=1</code>	<code>a=a+1</code>
<code>-=</code>	기존변수의 값에 값을 뺌.	<code>a-=1</code>	<code>a=a-1</code>
<code>*=</code>	기존 변수의 값에 값을 곱	<code>a*=1</code>	<code>a=a*1</code>
<code>/=</code>	기존 변수의 값에 값을 나눔	<code>a/=1</code>	<code>a=a/1</code>
<code>%=</code>	기존 변수의 값에 값을 구함	<code>a%=1</code>	<code>a=a%1</code>

- 쳐보는!!!

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  | // 변수를 선언합니다.
7  |  |  | let list = ''
8
9  |  |  | // 연산자를 사용합니다.
10 |  |  | list += '<ul>'
11 |  |  | list += '  <li>Hello</li>'
12 |  |  | list += '  <li>JavaScript..!</li>'
13 |  |  | list += '<ul>'
14
15 |  |  | // 문서에 출력합니다.
16 |  |  | document.write(list)
17 |  |  | </script>
18 |  |  | </head>
19 |  |  | <body>
20 |  |  | </body>
21 |  |  | </html>
22
```

- 복합 대입 연산자에 이어 변수와 함께 사용할 수 있는 연산자.
- 복합 대입 연산자를 간략하게 사용한 형태

증감연산자	설명
변수++	기존의 변수 값에 1을 더함.(후위)
++변수	기존의 변수 값에 1을 더함.(전위)
변수--	기존의 변수 값에 1을 뺌.(후위)
--변수	기존의 변수 값에 1을 뺌.(전위)

- 쳐보는!!!

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  // 변수를 선언합니다.
7  |  |  |  let number = 10
8
9
10 |  |  |  // 출력합니다.
11 |  |  |  alert(number++)
12 |  |  |  alert(number++)
13 |  |  |  alert(number++)
14 |  |  |  </script>
15 |  |  </head>
16 |  <body>
17 |  </body>
18 </html>
```

- 상수와 변수로 선언하지 않은 식별자의 자료형
- 종류
 - 상수와 변수로 선언하지 않은 식별자
 - > `typeof(abc)`
 - < 'undefined'
 - > `typeof(식별자)`
 - < 'undefined'
- 값이 없는 변수
 - > `let a`
 - < `undefined`
 - > `typeof(a)`
 - < 'undefined'

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white lab coat, suggesting a medical or scientific professional. The hands are positioned as if the person is interacting with the device. A large, semi-transparent white rectangular box is overlaid on the center-left portion of the image, containing the Korean text.

자료형 변환

- 문자열 자료형을 입력할 때 사용하는 함수 : prompt()

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  | // 상수를 선언합니다.
7  |  |  | const input = prompt('message', '_default')
8
9  |  |  | // 출력합니다.
10 |  |  | alert(input)
11 |  |  | </script>
12 |  |  | </head>
13 |  |  | <body>
14 |  |  | </body>
15 |  |  | </html>
16
```

- boolean 자료형을 입력할 때 사용하는 함수 : confirm()

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  | // 상수를 선언합니다.
7  |  |  | const input = confirm('수락하시겠습니까?')
8  |
9  |  |  | // 출력합니다.
10 |  |  | alert(input)
11 |  |  | </script>
12 |  |  | </head>
13 |  |  | <body>
14 |  |  | </body>
15 |  |  | </html>
16
```

- 다른 자료형을 숫자 자료형으로 변환할 때 사용하는 함수 : confirm()
- 숫자로 변환안될 경우: NaN

```
> Number("$223")
< NaN


---


> typeof(Number("$223"))
< 'number'
```

- 다른 자료형을 Boolean 자료형으로 변환할 때 사용하는 함수 : boolean()
- !! – 불 자료형으로 변환

```
> !!273
< true


---


> !!'안녕하세요'
< true


---


> !!}
< false
```

```
> Boolean(0)
< false


---


> Boolean(NaN)
< false


---


> Boolean('')
< false


---


> Boolean(null)
< false


---


> let 변수
< undefined


---


> Boolean(변수)
< false
```

- 사용자에게 inch로 값을 입력 받아서, cm단위로 가공하고 결과를 보여주는 프로그램을 만드시오.
- 변수는 rawInput(입력창), inch(받은 입력값), cm(변환된 값)
- 1 inch는 2.54cm
- 결과는 alert창으로 해주세요.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6              // 숫자를 입력받습니다.
7              const rawInput = prompt('inch 단위의 숫자를 입력해주세요.')
8
9              // 입력받은 데이터를 숫자형으로 변경하고 cm 단위로 변경합니다.
10             const inch = Number(rawInput)
11             const cm = inch * 2.54
12
13             // 출력합니다.
14             alert(` ${inch}inch는 ${cm}cm 입니다.`)
15         </script>
16     </head>
17     <body>
18     </body>
19 </html>
```

chapter 3

조건문



if

- if

```
if (boolean 표현식){
```

()안에 값이 참일 경우 실행해야 할 문장

```
}
```

- if else

```
if (boolean 표현식){
```

()안에 값이 참일 경우 실행해야 할 문장

```
} else {
```

()안에 값이 거짓일 경우 실행해야 할 문장

```
}
```

- 중첩

```
if (boolean 표현식1){  
    if (boolean 표현식2){  
        ()안에 2값이 참일 경우 실행해야 할 문장  
    } else {  
        ()안에 2값이 거짓일 경우 실행해야 할 문장  
    }  
} else {  
    if (boolean 표현식3){  
        ()안에 3값이 참일 경우 실행해야 할 문장  
    } else {  
        ()안에 3값이 거짓일 경우 실행해야 할 문장  
    }  
}
```

1값이 참일 경우

1값이 거짓일 경우

- if else if

```
if (boolean 표현식){  
    ()안에 값이 참일 경우 실행해야 할 문장  
} else if (boolean 표현식){  
    ()안에 값이 참일 경우 실행해야 할 문장  
} else if (boolean 표현식){  
    ()안에 값이 참일 경우 실행해야 할 문장  
} else {  
    ()안에 값이 거짓일 경우 실행해야 할 문장  
}
```

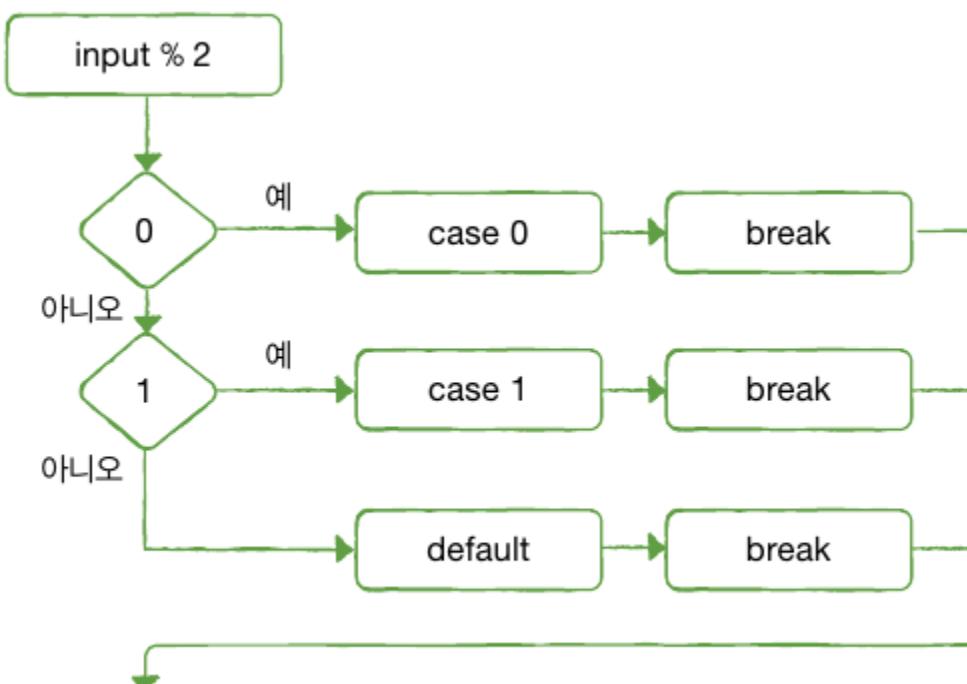
A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. A large, semi-transparent white rectangular box is overlaid on the center-left portion of the image. Inside this box, the word "switch" is written in a bold, black, sans-serif font.

switch

```
switch (조건) {  
    case 조건A:  
        break  
    case 조건B:  
        break  
    defalt:  
        break  
}
```

```
1  <!DOCTYPE html>  
2  <html>  
3      <head>  
4          <title></title>  
5          <script>  
6              // 변수를 선언합니다.  
7              const input = Number(prompt("숫자를 입력하세요.", "숫자"))  
8  
9              // 조건문  
10             switch (input % 2) {  
11                 case 0:  
12                     alert("짝수입니다.")  
13                     break  
14                 case 1:  
15                     alert("홀수입니다.")  
16                     break  
17                 default:  
18                     alert("숫자가 아닙니다.")  
19                     break  
20             }  
21             </script>  
22         </head>  
23         <body>  
24         </body>  
25     </html>
```

- break : switch 조건문이나 반복문을 빠져나가기 위해 사용하는 키워드. break를 만나면 키워드를 감싼 switch 조건문이나 반복문을 완전히 빠져나감.
- switch 조건문



- 조건문과 비슷한 역할을 하는 연산자

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  | // 변수를 선언합니다.
7  |  |  | const input = prompt('숫자를 입력해주세요.', '')
8  |  |  | const number = Number(input)
9
10 |  |  | // 조건문
11 |  |  | const result = (number >= 0 ? '0 이상의 숫자입니다.' : '0보다 작은 숫자입니다.')
12 |  |  | alert(result)
13 |  |  </script>
14 |  </head>
15 |  <body>
16 |  </body>
17 </html>
```

- 논리합 연산자 사용.
 - 불 표현 || 불표현식이 거짓일 때 실행할 문장.
 - true || ○○○
- 논리곱 연산자 사용.
 - 결과가 거짓인 불 표현식 && 불 표현식이 참일 때 실행할 문장
 - false && ○○○

- 사용자에게 출생연도로 값을 입력 받아서, 띠를 보여주는 프로그램을 만드시오.
- 변수는 rawInput(입력창), year(받은 입력값), e(12로 나눈값), result(띠)
- 결과는 alert창으로 해주세요. (" 1973년에 태어났다면, 소띠입니다. ")

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <script>
      const rawInput = prompt('태어난 해를 입력해주세요.', '')
      const year = Number(rawInput)
      const e = year % 12

      let result
      if (e === 0) { result = '원숭이' }
      else if (e === 1) { result = '닭' }
      else if (e === 2) { result = '개' }
      else if (e === 3) { result = '돼지' }
      else if (e === 4) { result = '쥐' }
      else if (e === 5) { result = '소' } Loading...
      else if (e === 6) { result = '호랑이' }
      else if (e === 7) { result = '토끼' }
      else if (e === 8) { result = '용' }
      else if (e === 9) { result = '뱀' }
      else if (e === 10) { result = '말' }
      else if (e === 11) { result = '양' }
      alert(` ${year}년에 태어났다면 ${result} 땅입니다.`)
    </script>
  </head>
  <body></body>
</html>
```

chapter 4

반복문

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. A large, semi-transparent white rectangular box covers the center of the image, containing the Korean text "백일".

백일

- 여러 개의 변수를 한번에 선언해 다룰 수 있는 자료형.
- [요소, 요소, 요소, 요소, ...]

```
> const array = [273, 'String', true,  
function(){}, {}, [273,103]]  
< undefined  
  
> array  
< (6) [273, 'String', true, f, {...}, A  
rray(2)] i  
  0: 273  
  1: "String"  
  2: true  
  ▶ 3: f ()  
  ▶ 4: {}  
  ▶ 5: (2) [273, 103]  
  length: 6  
  ▶ [[Prototype]]: Array(0)
```

- 인덱스 : 요소의 순서
- 0부터 시작
- 배열[인덱스]

```
> const array = [273, 'String', true,
  function(){}
, {}, [273,103]]
< undefined


---


> array
< ▶ (6) [273, 'String', true, f, ...],
  Array(2)] i
    0: 273
    1: "String"
    2: true
    ▶ 3: f ()
    ▶ 4: {}
    ▶ 5: (2) [273, 103]
    length: 6
    ▶ [[Prototype]]: Array(0)


---


> array[2]
< true
```

- 배열의 길이 : 배열.length
- 배열에 추가 : 배열.push(요소)
- 배열에 삭제
 - 인덱스를 기반으로 제거 : splice()
 - 값을 기반으로 제거 : indexOf()

```

> array.length
< 6


---


> array.push('soo')
< 7


---


> array
< ▶ (7) [273, 'String', true, f, {...}, Array(2), 'soo']


---


> array.splice(2,1)
< ▶ [true]


---


> array
< ▶ (6) [273, 'String', f, {...}, Array(2), 'soo']


---


> array.indexOf('String')
< 1


---


> array.splice(1,1)
< ▶ ['String']


---


> array
< ▶ (5) [273, f, {...}, Array(2), 'soo']

```

- 특정위치 추가할 때 : splice()
- 배열.splice(인덱스, 0, 요소)

```
> array
< ▶ (5) [273, f, {...}, Array(2), 'soo']
> array.splice(1, 0, "양파")
< ▶ []
> array
< (6) [273, '양파', f, {...}, Array(2), ...]
```

- 자료처리를 위해서 다양한 연산자, 함수, 메소드를 제공
- 처리 후 원본의 상태변화에 따라 2가지로 구분
 - 비파괴적 처리 : 원본은 변경이 없이 데이터가 처리 되는 경우
 - 파괴적 처리 : 원본 자체의 변경으로 데이터 처리가 되는 경우
- 과거의 컴퓨터 메모리는 저장공간이 매우 부족
- 이에 따라 메모리 공간에 대한 효율적인 관리가 필요.
- 따라서 배열 같은 자료는 메모리 절감을 위해서 파괴적 처리

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. The background is blurred.

반복문

- 배열과 같이 사용할 수 있는 반복문.
- 배열 요소를 하나하나 꺼내서 특정 문장을 실행할 때 사용.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  const todos = ['우유구매', '업무 메일 확인하기', '필라테스 수업']
7
8  |  |  |  for (const i in todos) {
9  |  |  |  |  console.log(` ${i}번째 할 일: ${todos[i]}`)
10 |  |  |  }
11 |  |  |  </script>
12 |  |  |  </head>
13 |  |  |  <body></body>
14 |  |  |  </html>
```

- 인덱스는 항상 불안한 요소.
- 안정성을 위해서 요소의 값을 반복할 때 안정적으로 사용을 위해서 for of 반복문 사용.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  const todos = ['우유구매', '업무 메일 확인하기', '필라테스 수업']
7  |  |  |  for (const todo of todos) {
8  |  |  |  |  console.log(`오늘의 할 일: ${todo}`)
9  |  |  |  }
10 |  |  |  </script>
11 |  |  </head>
12 |  |  <body></body>
13 |  </html>
```

- 일반적인 반복문. 범용적으로 사용

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6              for (let i = 0; i < 5; i++) {
7                  console.log(`#${i}번째 반복입니다.`)
8              }
9          </script>
10     </head>
11     <body></body>
12 </html>
```

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  const todos = ['우유 구매', '업무 메일 확인하기', '필라테스 수업']
7
8  |  |  |  for (let i = 0; i < todos.length; i++) {
9  |  |  |  |  console.log(`#${i}번째 할 일: ${todos[i]}`)
10 |  |  |  }
11 |  |  </script>
12 |  </head>
13 |  <body></body>
14 </html>
```

- 불표현식이 true이면 계속해서 문장을 실행함.
- 위험 할 수 있음.(무한루프!!!)
- 실행금지

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6              let i = 0
7              while (true) {
8                  alert(`#${i}번째 반복입니다.`)
9                  i = i+1
10             }
11         </script>
12     </head>
13     <body></body>
14 </html>
```

- while 문과 for 문은 서로 대체해서 사용이 가능.
- 배열을 쓸 경우는 .length등으로 범위를 한정하고 끝나도록 하는 게 중요.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  let i = 0
7  |  |  |  const array = [1, 2, 3, 4, 5]
8
9  |  |  |  while (i < array.length) {
10 |  |  |  |  console.log(` ${i} : ${array[i]}`)
11 |  |  |  |  i++
12 |  |  |  }
13 |  |  |  </script>
14 |  |  </head>
15 |  <body></body>
16 </html>
```

- 묻고 break를 이용해서 나갈 수 있도록 조치.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  | // 반복문
7  |  |  | for (let i = 0; true; i++) {
8  |  |  | | alert(i + '번째 반복문입니다.')
9  |
10 |  |  | // 진행 여부를 물어봅니다.
11 |  |  | const isContinue = confirm('계속하시겠습니까?')
12 |  |  | if (!isContinue) {
13 |  |  | | break
14 |  |  | }
15 |  |
16 |  | // 프로그램의 종료를 확인합니다.
17 |  | alert('프로그램 종료')
18 |  | </script>
19 |  | </head>
20 |  | <body></body>
21 |  |
22 |  </html>
```

- 반복작업을 멈추고, 처음으로 돌아가서 다음 반복작업을 진행.
- 조건문을 사용해서 홀수일 때는 continue 키워드를 만나 바로 다음 반복작업으로 넘어가므로 짹수 합만 구해짐.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // let 변수를 선언합니다.
7       let output = 0
8
9       // 반복문
10      for (let i = 1; i <= 10; i++) {
11        // 조건문
12        if (i % 2 === 1) {
13          // 홀수면 현재 반복을 중지하고 다음 반복을 수행합니다.
14          continue
15        }
16        output += i
17      }
18
19      // 출력합니다.
20      alert(output)
21    </script>
22  </head>
23  <body></body>
24 </html>
```

출처: 혼자 공부하는 자바스크립트

- for문을 2개 중첩해서, 좌측의 별표 트리를 만들시요.

*

**

- 이중 중첩으로 구문도 해보면 좋음.
- 외부 구문은 반복의 scope를 제어.
- 내부 구문은 실행 횟수를 제어.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  // let 변수를 선언합니다.
7  |  |  |  let output = ''
8
9
10 |  |  |  // 중첩 반복문
11 |  |  |  for (let i = 1; i < 10; i++) {
12 |  |  |  |  for (let j = 0; j < i; j++) {
13 |  |  |  |  |  output += '*'
14 |  |  |  |  }
15 |  |  |  |  output += '\n'
16 |  |  |  |
17 |  |  |  |  // 출력합니다.
18 |  |  |  |  console.log(output)
19 |  |  |  </script>
20 |  |  </head>
21 |  <body></body>
22 </html>
```

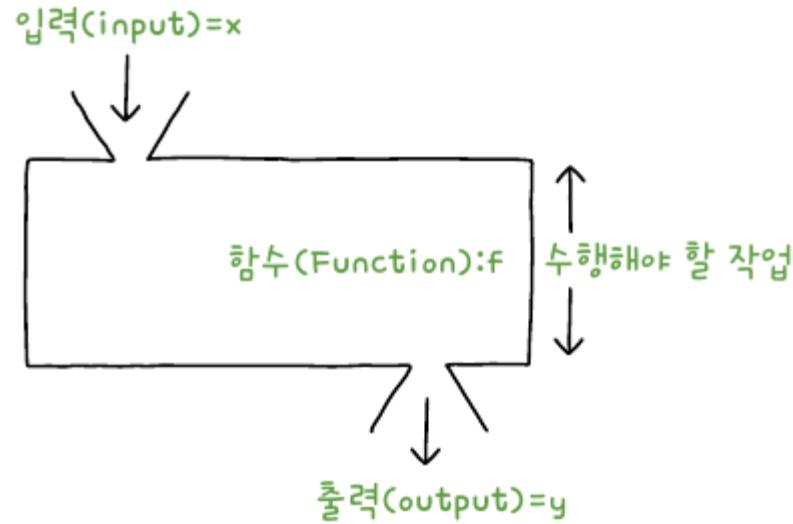
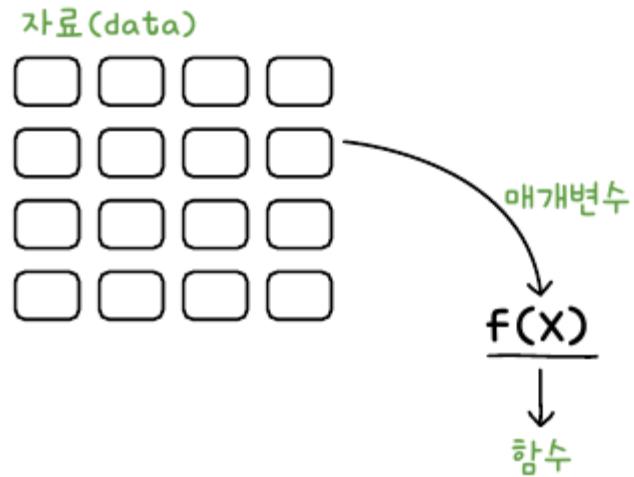
chapter 5

함수

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. The background is blurred.

함수 기본

- input값을 매개변수.
- output값을 리턴값.



- function() { }
- 이름이 붙지 않은 함수.

#함수 사용의 장점

- 반복되는 코드를 한번만 정의. 필요할 때는 호출해서 반복작업을 줄임.
- 긴 프로그램을 기능별로 나눠 여러 함수로 작성 모듈화로 가독성을 높임.
- 기능별(함수별)로 수정이 가능해 유지보수가 쉽다.

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 변수를 선언합니다.
7        const 함수 = function () {
8          console.log('함수 내부의 코드입니다 ... 1')
9          console.log('함수 내부의 코드입니다 ... 2')
10         console.log('함수 내부의 코드입니다 ... 3')
11         console.log('')
12     }
13
14    // 함수를 호출합니다.
15    함수()
16    함수()
17
18    // 출력합니다.
19    console.log(typeof 함수)
20    console.log(함수)
21    </script>
22  </head>
23  <body></body>
24 </html>

```

함수 내부의 코드입니다 ... 1	5-1-1.html:8
함수 내부의 코드입니다 ... 2	5-1-1.html:9
함수 내부의 코드입니다 ... 3	5-1-1.html:10
함수 내부의 코드입니다 ... 1	5-1-1.html:11
함수 내부의 코드입니다 ... 2	5-1-1.html:12
함수 내부의 코드입니다 ... 3	5-1-1.html:13
function	5-1-1.html:14
f () {	5-1-1.html:15
console.log('함수 내부의 코드입니다 ... 1')	
console.log('함수 내부의 코드입니다 ... 2')	
console.log('함수 내부의 코드입니다 ... 3')	
console.log('')	



- 일반적인 함수 생성

```
function 함수() {  
}
```

- 선언적 함수 생성

```
let 함수 = function(){  
}
```

- 매개변수 : 함수를 호출할 때 괄호안에 적는 것
 - prompt()함수를 사용할 때 매개변수로 message를 넣어야 함.
- 리턴값 : 함수의 최종 결과
- 일반적인 함수의 형태

```
function 함수( 매개변수 ) {  
    let output = 초기값  
    처리~~  
    return output  
}
```

함수 살펴 보기.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6              function min(array) {
7                  let output = array[0]
8                  for (const item of array) {
9                      // 현재 output 보다 더 작은 item이 있다면
10                     if (output > item) {
11                         // output 값을 item으로 변경
12                         output = item
13                     }
14                 }
15                 return output
16             }
17
18             const testArray = [52, 273, 32, 103, 275, 24, 57]
19             console.log(` ${testArray} 중에서 `)
20             console.log(` 최솟값 = ${min(testArray)} `)
21         </script>
22     </head>
23     <body></body>
24 </html>
```

- function (... 나머지 매개변수)
- 가변 매개변수 : 호출할 때 매개변수의 개수가 고정적이지 않은 함수.
- 자바스크립트에서 이러한 함수를 구현할 때는 나머지 매개변수라는 특이한 형태의 문법을 사용.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        function sample(a,b, ...c) {
7          console.log(a, b, c)
8        }
9        sample(1, 2)
10       sample(1, 2, 3)
11       sample(1, 2, 3, 4)
12     </script>
13   </head>
14   <body></body>
15 </html>
```

```
1 2 ▶ Array(0) ⓘ
  length: 0
  ► [[Prototype]]: Array(0)
1 2 ▶ Array(1) ⓘ
  0: 3
  length: 1
  ► [[Prototype]]: Array(0)
1 2 ▶ Array(2) ⓘ
  0: 3
  1: 4
  length: 2
  ► [[Prototype]]: Array(0)
```

- 매개변수로 숫자나 특정 데이터 타입으로만 받을 수 있을 경우 배열요소를 하나씩 전개해서 입력하는 방법만 가능.
- 자바스크립트에서는 배열을 전개해서 함수의 매개변수로 전달 해주는 연산자가 있음.
- 함수 이름(...배열)

```
1  <!DOCTYPE html>
2  <html>
3  | <head>
4  | | <title></title>
5  | | <script>
6  | | | // 단순하게 매개변수를 모두 출력하는 함수
7  | | | function sample(...items) {
8  | | | | console.log(items)
9  | | |
10 | | | // 전개 연산자 사용 여부 비교하기
11 | | | const array = [1, 2, 3, 4]
12 |
13 |
14 | | console.log('# 전개 연산자를 사용하지 않은 경우')
15 | | sample(array)
16 | | console.log('# 전개 연산자를 사용한 경우')
17 | | sample(...array)
18 | </script>
19 | </head>
20 | <body></body>
21 | </html>
```

전개 연산자를 사용하지 않은 경우

▼ Array(1) i

▶ 0: (4) [1, 2, 3, 4]

length: 1

▶ [[Prototype]]: Array(0)

전개 연산자를 사용한 경우

▼ Array(4) i

0: 1

1: 2

2: 3

3: 4

length: 4

▶ [[Prototype]]: Array(0)

- 매개변수 기본 입력값을 지정할 때 사용.
- 함수 이름(매개변수, 매개변수=기본값, 매개변수=기본값)

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  function earnings (name, wage=8590, hours=40) {
7  |  |  |  |  console.log(`# ${name} 님의 급여 정보`)
8  |  |  |  |  console.log(`- 시급: ${wage}원`)
9  |  |  |  |  console.log(`- 근무 시간: ${hours}시간`)
10 |  |  |  |  console.log(`- 급여: ${wage * hours}원`)
11 |  |  |  |  console.log('')
12 |  |  |
13 |  |  // 최저 임금으로 최대한 일하는 경우
14 |  |  earnings('구름')
15 |
16 |
17 |  |  // 시급 1만원으로 최대한 일하는 경우
18 |  |  earnings('별', 10000)
19 |
20 |  |  // 시급 1만원으로 52시간 일한 경우
21 |  |  earnings('인성', 10000, 52)
22 |  |</script>
23 |  </head>
24 |  <body></body>
25 </html>
```

구름 님의 급여 정보

- 시급: 8590원
- 근무 시간: 40시간
- 급여: 343600원

별 님의 급여 정보

- 시급: 10000원
- 근무 시간: 40시간
- 급여: 400000원

인성 님의 급여 정보

- 시급: 10000원
- 근무 시간: 52시간
- 급여: 520000원

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. A large, semi-transparent white rectangular box is overlaid on the center-left portion of the image. Inside this box, the Korean text "함수 고급" is displayed in a bold, black, sans-serif font.

함수 고급

- 자바스크립트에서는 “함수도 하나의 자료”
- 2010년 전후로 비동기 프로그래밍이 각광 받으며, 익명함수의 문법적 가치를 인정받음.
- 람다 또는 익명함수라는 게 기본문법에 많이 포함됨.
- 함수의 매개변수에 자료형 뿐만 아니라 함수도 전달이 가능.

- 매개변수로 전달하는 함수.

```

<script>
  // 함수를 선언합니다.
  function callThreeTimes (callback) {
    for (let i = 0; i < 3; i++) {
      callback(i)
    }
  }

  // 함수를 호출합니다.
  callThreeTimes(function (i) {
    console.log(`#${i}번째 함수 호출`)
  })
</script>

```

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 함수를 선언합니다.
7        function callThreeTimes (callback) {
8          for (let i = 0; i < 3; i++) {
9            callback(i)
10           }
11         }
12
13        function print (i) {
14          console.log(`#${i}번째 함수 호출`)
15        }
16
17        // 함수를 호출합니다.
18        callThreeTimes(print)
19      </script>
20    </head>
21    <body></body>
22  </html>

```

- function (value, index, array){ } 형태의 콜백함수를 사용.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  const numbers = [273, 52, 103, 32, 57]
7
8  |  |  |  numbers.forEach(function (value, index, array) {
9  |  |  |  |  console.log(` ${index} 번째 요소 : ${value}` )
10 |  |  |  })
11 |  |  </script>
12 |  </head>
13 |  <body></body>
14 </html>
```

- function (value, index, array){ } 형태의 콜백함수를 사용.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  // 배열을 선언합니다.
7  |  |  |  let numbers = [273, 52, 103, 32, 57]
8
9
10 |  |  // 배열의 모든 값을 제곱합니다.
11 |  |  numbers = numbers.map(function (value, index, array) {
12 |  |  |  return value * value
13 |  |  })
14
15 |  |  // 출력합니다.
16 |  |  numbers.forEach(console.log)
17 |  |  </script>
18 |  </head>
19 |  <body></body>
20 </html>
```

- 콜백함수에서 리턴하는 값이 true인 것들만 모아서 새로운 배열을 만드는 함수.

원래 배열: 0,1,2,3,4,5

짝수만 추출: 0,2,4

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       const numbers = [0, 1, 2, 3, 4, 5]
7       const evenNumbers = numbers.filter(function (value) {
8         return value % 2 === 0
9       })
10
11      console.log(`원래 배열: ${numbers}`)
12      console.log(`짝수만 추출: ${evenNumbers}`)
13    </script>
14  </head>
15  <body></body>
16 </html>
```

- map(), filter() 함수처럼 단순한 형태의 콜백 함수를 쉽게 입력하고자 하는 함수 생성 방법.
- function 키워드 대신 화살표(=>)를 사용.
- (매개변수) => 리턴값.

```
> const array = [0,1,2,3,4,5,6,7,8,9]
< undefined


---


> array.map((value) => value * value)
< ▶ (10) [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- 메소드가 리턴하는 값을 기반으로 해서 함수를 줄줄이 사용하는 것.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 배열을 선언합니다.
7        let numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
8
9        // 배열의 메소드를 연속적으로 사용합니다.
10       numbers
11         .filter((value) => value % 2 === 0)
12         .map((value) => value * value)
13         .forEach((value) => {
14           console.log(value)
15         })
16       </script>
17     </head>
18     <body></body>
19   </html>
```

0

4

16

36

64

- 특정 시간마다 또는 특정 시간 이후 콜백함수를 호출할 수 있는 함수.
 - setTimeout(함수, 시간) : 특정 시간 후에 함수를 한번 호출
 - setInterval(함수, 시간) : 특정 시간마다 함수를 호출

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       setTimeout(() => {
7         console.log(`1초 후에 실행됩니다.`)
8       }, 1 * 1000)
9
10    let count = 0
11    setInterval(() => {
12      console.log(`1초마다 실행됩니다.(${count}번째)`)
13      count++
14    }, 1 * 1000)
15  </script>
16 </head>
17 <body></body>
18 </html>
```

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <!-- 다른 곳에서 가져온 자바스크립트 코드 -->
6          <script>
7              let pi = 3.14
8              console.log(`파이 값은 ${pi}입니다.`)
9          </script>
10
11         <!-- 내가 만든 자바스크립트 코드 -->
12         <script>
13             let pi = 3.141592
14             console.log(`파이 값은 ${pi}입니다.`)
15         </script>
16     </head>
17     <body></body>
18 </html>

```

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <!-- 다른 곳에서 가져온 자바스크립트 코드 -->
6          <script>
7              let pi = 3.14
8              console.log(`파이 값은 ${pi}입니다.`)
9
10         // 블록을 사용한 스코프 생성
11         {
12             let pi = 3.141592
13             console.log(`파이 값은 ${pi}입니다.`)
14         }
15         console.log(`파이 값은 ${pi}입니다.`)
16
17         // 함수 블록을 사용한 스코프 생성
18         function sample() {
19             let pi = 3.141592
20             console.log(`파이 값은 ${pi}입니다.`)
21         }
22
23         sample()
24         console.log(`파이 값은 ${pi}입니다.`)
25     </script>
26     </head>
27     <body></body>
28 </html>

```

- while 문 : 조건을 중심으로 반복할 때.
- for 문 : 횟수를 중심으로 또는 배열들을 중심으로 반복할 때.
- 익명함수와 선언적 함수는 사용하는 상황이 비슷.
- 최근 추세는 익명함수를 좀더 선호.

- 순차적인 코드 실행에서 코드가 해당 줄을 읽을 때 생성.
- 2번째가 출력.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 변수를 선언합니다.
7       let 익명함수
8
9       // 익명 함수를 2번 생성합니다.
10      익명함수 = function () {
11        console.log('1번째 익명 함수입니다.')
12      }
13      익명함수 = function () {
14        console.log('2번째 익명 함수입니다.')
15      }
16
17      // 익명 함수를 호출합니다.
18      익명함수()
19    </script>
20  </head>
21  <body></body>
22 </html>
```

- 순차적인 코드 실행이 일어나기 전에 생성.
- 같은 블록이라면, 어디에서 함수를 호출해도 동일.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 선언적 함수를 호출합니다.
7        선언적함수()
8
9
10   // 선언적 함수를 2번 생성합니다.
11   function 선언적함수 () {
12     | console.log('1번째 선언적 함수입니다.')
13   }
14   function 선언적함수 () {
15     | console.log('2번째 선언적 함수입니다.')
16   }
17   </script>
18   </head>
19   <body></body>
20 </html>
```

- 선언적 함수는 먼저 생성되고, 이후에 순차적인 코드 진행을 시작하면서 익명함수를 생성.
- 이 경우 코드의 순서와 관계 없이 “익명 함수입니다”

```
1  <!DOCTYPE html>
2  <html>
3  |   <head>
4  |   |   <title></title>
5  |   |   <script>
6  |   |   |   // 익명 함수를 2번 생성합니다.
7  |   |   |   함수 = function () {
8  |   |   |   |   console.log('익명 함수입니다.')
9  |   |   |   }
10 |
11 // 선언적 함수를 2번 생성하고 할당합니다.
12 function 함수 () {
13 |   console.log('선언적 함수입니다.')
14 }
15
16 // 함수를 호출합니다.
17 함수()
18 </script>
19 </head>
20 <body></body>
21 </html>
```

- let을 사용해서 함수를 선언할 경우 덮어 쓰는 것을 방지하고, Uncaught SyntaxError를 내서 원천 차단.

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  // 익명 함수를 2번 생성합니다.
7  |  |  |  let 함수 = function () {
8  |  |  |  |  console.log('익명 함수입니다.')
9  |  |  |  }
10 |
11 |  // 선언적 함수를 2번 생성하고 할당합니다.
12 |  function 함수 () {
13 |  |  console.log('선언적 함수입니다.')
14 |  }
15 |
16 |  // 함수를 호출합니다.
17 |  함수()
18 |  </script>
19 |  </head>
20 |  <body></body>
21 |  </html>
```

chapter 6

액체

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. The background is blurred.

객체 기본

- 객체란 추상적 의미
- 실제로 존재하는 사물에 대한 메타포어.
- 속성과 동작(역시 속성)을 표현하는 메소드로 구성.
- 배열도 객체.
- 객체는 배열이 인덱스를 썼다면, key를 사용.

```
const product {          객체  
  제품명 : '건조 망고',  
  유형 : '당절임',  
  성분 : '망고 '  
}
```

```
const product {  
    제품명 : '건조 망고',  
    유형 : '당절임',  
    성분 : '망고'  
}
```

객체에 접근

대괄호	온점
Product['제품명']	Product.제품명
Product['유형']	Product.유형
Product['성분']	Product.성분

- 배열 내부에 있는 값을 "요소" 라고 함.
- 객체 내부에 있는 값을 "속성" 이라고 함.
- 배열에서 처럼 모든 형태의 자료형을 속성으로 가질 수 있음.
- 객체 속성 중 함수 자료형인 속성을 특별히 메소드라고 함.

- 메소드 내에서 자기 자신이 가진 속성을 표시하고 싶을 때는 자신이 가진 속성임을 분명하게 표시해야 함.
- 자기자신의 속성임을 표시할 때, this

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6              // 변수를 선언합니다.
7              const pet = {
8                  name: '구름',
9                  eat: function (food) {
10                      alert(this.name + '은/는 ' + food + '을/를 먹습니다.')
11                  }
12              }
13
14              // 메소드를 호출합니다.
15              pet.eat('밥')
16          </script>
17      </head>
18      <body></body>
19  </html>
```

출처: 혼자 공부하는 자바스크립트

```

1  <!DOCTYPE html>
2  <html>
3  | <head>
4  | | <title></title>
5  | | <script>
6  | | | // 객체를 선언합니다.
7  | | | const student = {}
8  | | | student.이름 = '윤인성'
9  | | | student.취미 = '악기'
10 | | | student.장래희망 = '생명공학자'
11
12 // 출력합니다.
13 console.log(JSON.stringify(student, null, 2))
14 </script>
15 </head>
16 <body></body>
17 </html>

```

```

1  <!DOCTYPE html>
2  <html>
3  | <head>
4  | | <title></title>
5  | | <script>
6  | | | // 객체를 선언합니다.
7  | | | const student = {}
8  | | | student.이름 = '윤인성'
9  | | | student.취미 = '악기'
10 | | | student.장래희망 = '생명공학자'
11
12 // 객체의 속성을 제거합니다.
13 delete student.장래희망
14
15 // 출력합니다.
16 console.log(JSON.stringify(student, null, 2))
17 </script>
18 </head>
19 <body></body>
20 </html>

```

- 객체내에 바로 선언 가능.

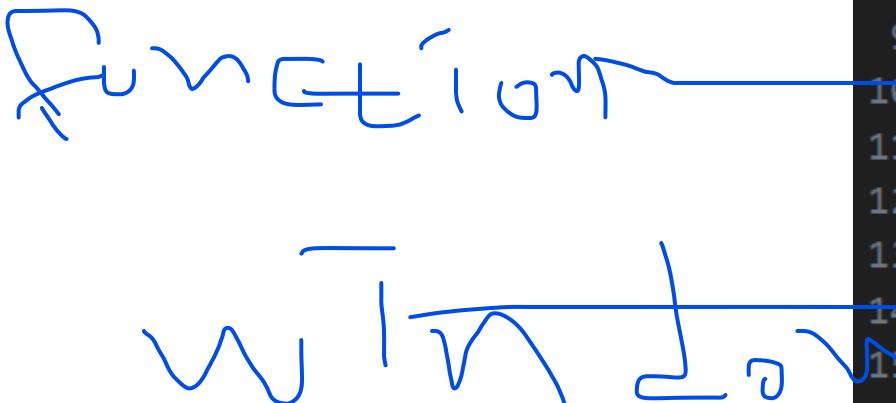
```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  | // 객체를 선언합니다.
7  |  |  | const pet = {
8  |  |  | | name: '구름',
9  |  |  | | eat (food) {
10 |  |  | | | alert(this.name + '은/는 ' + food + '을/를 먹습니다.')
11 |  |  | |
12 |  |  |
13 |  |  | // 메소드를 호출합니다.
14 |  |  | pet.eat('밥')
15 |  |  |
16 |  |  | </script>
17 |  |  |
18 |  |  | <body></body>
19 |  |  | </html>
```

- `function(){ }` 형태의 익명함수와 `() =>{ }` 형태로 선언하는 화살표 함수는 객체의 메소드로 사용될 때 `this` 키워드를 다루는 방식이 차이가 있음.

▼ Object i

- ▶ a: f ()
- ▶ b: () => { `console.log(this)` }
- ▶ [[Prototype]]: Object

▶ Window



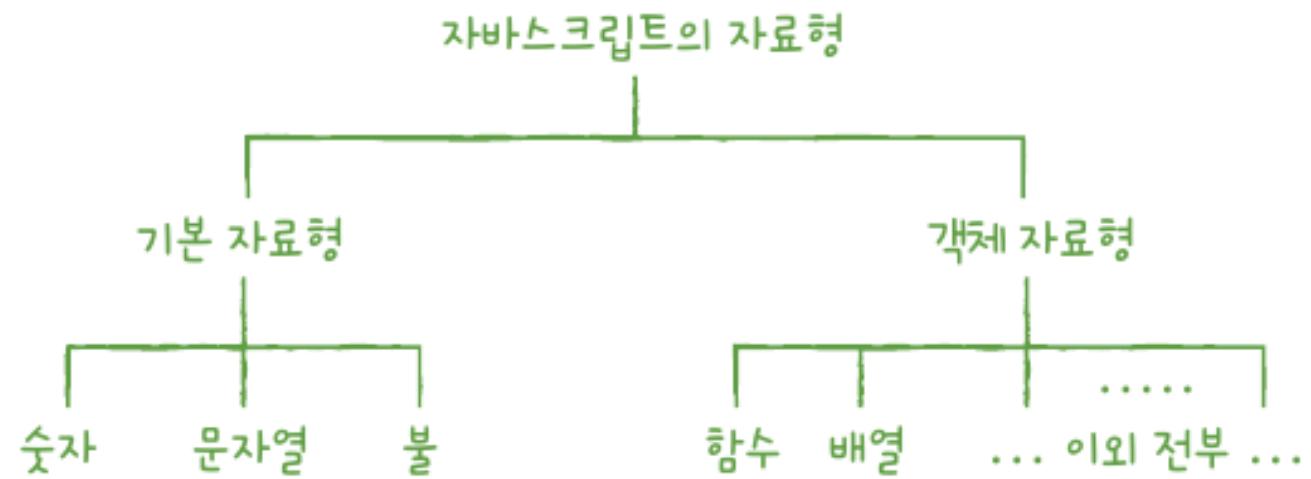
```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 변수를 선언합니다.
7      const test = {
8        a: function () {
9          console.log(this)
10       },
11        b: () => {
12          console.log(this)
13       }
14     }
15     // 메소드를 호출합니다.
16     test.a()
17     test.b()
18   </script>
19 </head>
20 <body></body>
21
22 </html>

```

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. The background is blurred.

속성과 메소드



- 속성과 메소드를 가질 수 있는 모든 것이 객체.
- `typeof` 연산자를 통해 배열의 자료형을 확인하면 'object'로 나옴. (`.isArray()`로 배열인지 확인)
- 자바스크립트에서 함수는 '일급 객체'

배열도 객체

```
> const a = []
< undefined
> a.sample = 10
< 10
> a.sample
< 10
```

함수도 객체

```
> function b () {
}
< undefined
> b.sample = 10
< 10
> b.sample
< 10
```

- 숫자, 문자열, 불
- 속성을 가질 수 없음.
- 따라서 객체의 속성 접근 방법으로 접근할 수 없음.
- 다만 아래와 같은 기본자료형을 객체로 변환해주는 함수가 있음. (new 로 선언해줘야 객체임. 안할 경우 기본 자료형.)
 - Number()
 - String()
 - Boolean()

- 숫자 객체 전체에 어떤 속성과 메소드를 추가할 수 있다면 기본 자료형 숫자도 속성과 메소드를 사용 가능.
- 어떤 객체에 prototype이라는 속성이 바로 객체에 속성과 메소드를 추가시킬 때 사용하는 속성

```
> Number.prototype.sample = 10
< 10
> const i = 273
< undefined
> i.sample
< 10
```

- `toFixed()` : 소수점 이하 몇 자리 까지만 출력하고 싶을 때 사용.
소수점 아래 2자리까지 출력-`toFixed(2)`
- `isNaN()` : 어떤 숫자가 NaN인지, 또는 Infinity인지 확인
- `isFinite()` : 어떤 숫자가 양의 무한대 숫자와 음의 무한대 숫자인지 여부 확인.

- trim() : 문자열 앞뒤 공백(띄어쓰기, 줄바꿈)을 제거
- split() : 매개변수로 준 문자열로 잘라서 배열로 만들어 리턴
- length() : 문자열의 길이
- indexOf() : 특정 인덱스에 문자열 리턴.

- 인터넷 상에 가장 많이 사용되는 자료 표현 형식.
- JavaScript Object Notation
- API의 상당수가 JSON으로 리턴을 해 줌
- 규칙
 - 값을 표현할 때는 문자열, 숫자, 불 자료형만 사용 가능(함수는 사용불가)
 - 문자열은 반드시 큰 따옴표로 만들어 사용.
 - key에도 반드시 따옴표 사용.
- JSON.stringify() : 자바스크립트 객체를 JSON 문자열로 변환
- JSON.parse() : JSON문자열을 자바스크립트 객체로 전개

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 자료를 생성합니다.
7       const data = [
8         {
9           name: '혼자 공부하는 파이썬',
10          price: 18000,
11          publisher: '한빛미디어'
12        },
13        {
14          name: 'HTML5 웹 프로그래밍 입문',
15          price: 26000,
16          publisher: '한빛아카데미'
17        }
18
19       // 자료를 JSON으로 변환합니다.
20       console.log(JSON.stringify(data))
21       console.log(JSON.stringify(data, null, 2))
22     </script>
23   </head>
24   <body></body>
25 </html>
```

[{"name": "혼자 공부하는 파이썬", "price": 18000, "publisher": "한빛미디어"}, {"name": "HTML5 웹 프로그래밍 입문", "price": 26000, "publisher": "한빛아카데미"}]
[
 {
 "name": "혼자 공부하는 파이썬",
 "price": 18000,
 "publisher": "한빛미디어"
 },
 {
 "name": "HTML5 웹 프로그래밍 입문",
 "price": 26000,
 "publisher": "한빛아카데미"
 }]
[6-2-3.html:18](#)
[6-2-3.html:19](#)

- 수학과 관련된 기본적인 연산에 사용.
- pi, e와 같은 수학 상수가 있음.
- Math.sin()
- Math.cos()
- Math.tan()
- Math.random() ← 난수반사법

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title></title>
5  <script>
6      const num = Math.random()
7
8      console.log('# 랜덤한 숫자')
9      console.log('0~1 사이의 랜덤한 숫자:', num)
10     console.log('')
11
12     console.log('# 랜덤한 숫자 범위 확대')
13     console.log('0~10 사이의 랜덤한 숫자:', num * 10)
14     console.log('0~50 사이의 랜덤한 숫자:', num * 50)
15     console.log('')
16
17     console.log('# 랜덤한 숫자 범위 이동')
18     console.log('-5~5 사이의 랜덤한 숫자:', num * 10 - 5)
19     console.log('-25~25 사이의 랜덤한 숫자:', num * 50 - 25)
20     console.log('')
21
22     console.log('# 랜덤한 정수 숫자')
23     console.log('-5~5 사이의 랜덤한 정수 숫자:', Math.floor(num * 1))
24     console.log('-25~25 사이의 랜덤한 정수 숫자:', Math.floor(num * 5))
25
26  </script>
27  </head>
28  <body></body>

```

랜덤한 숫자

0~1 사이의 랜덤한 숫자:

0.5415670414524547

랜덤한 숫자 범위 확대

0~10 사이의 랜덤한 숫자:

5.415670414524547

0~50 사이의 랜덤한 숫자:

27.078352072622735

랜덤한 숫자 범위 이동

-5~5 사이의 랜덤한 숫자:

0.41567041452454667

-25~25 사이의 랜덤한 숫자:

2.078352072622735

랜덤한 정수 숫자

-5~5 사이의 랜덤한 정수 숫자: 0

-25~25 사이의 랜덤한 정수 숫자: 2

```
06 > <> main.html
1   <!DOCTYPE html>
2   <html>
3     <head>
4       <title></title>
5       <script src="test.js"></script>
6       <script>
7         console.log('# main.html의 script 태그')
8         console.log('sample 값:', sample)
9       </script>
10      </head>
11      <body></body>
12    </html>
```

```
06 > JS test.js
1   console.log('# test.js 파일')
2   const sample = 10
```

test.js 파일
main.html의 script 태그
sample 값: 10

- 알아야 할 두가지.
 - CDN이란 콘텐츠 전송 네트워크(무척 빠르다.)
 - min 버전이란 자바스크립트를 zipping 한 파일.
- 설치

```
<script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
```

- 사용 이유
 - 브라우저에서 지원하지 않는 성능이 보장되어 있는 다양한 메소드를 가지고 있음.
 - 퍼포먼스 측면에서 native보다 더 나은 성능을 가짐.
 - npm이나 기타 패키지 매니저를 통해 쉽게 사용 가능.

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. The background is blurred.

객체/배열 고급

- undefined 인지 여부를 가지고 확인

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title></title>
5  <script>
6  // 객체를 생성합니다.
7  const object = {
8  name: '혼자 공부하는 파이썬',
9  price: 18000,
10 publisher: '한빛미디어'
11 }
12
13 // 객체의 기본 속성을 지정합니다.
14 object.name = object.name !== undefined ? object.name : '제목 미정'
15 object.author = object.author !== undefined ? object.author : '저자 미상'
16
17 // 객체를 출력합니다.
18 console.log(JSON.stringify(object, null, 2))
19 </script>
20 </head>
21 <body></body>
22 </html>
```

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title></title>
5  <script>
6  // 객체를 생성합니다.
7  const object = {
8  name: '혼자 공부하는 파이썬',
9  price: 18000,
10 publisher: '한빛미디어'
11 }
12
13 // 객체 내부에 속성이 있는지 확인합니다.
14 if (object.name !== undefined) {
15   console.log('name 속성이 있습니다.')
16 } else {
17   console.log('name 속성이 없습니다.')
18 }
19 if (object.author !== undefined) {
20   console.log('author 속성이 있습니다.')
21 } else {
22   console.log('author 속성이 없습니다.')
23 }
24 </script>
25 </head>
26 <body></body>
27 </html>
```

- [식별자, 식별자, 식별자, ..] = 배열
- 할당 연산자 왼쪽에 식별자(변수 또는 상수)의 배열을 넣고, 오른쪽에 배열을 위치시키면 배열 위치에 맞게 값들이 할당.
- 배열의 크기는 같을 필요는 없고, let이나 const 키워드로 사용 가능.

- [식별자, 식별자, 식별자, ..] = 배열
- 할당 연산자 왼쪽에 식별자(변수 또는 상수)의 배열을 넣고, 오른쪽에 배열을 위치시키면 배열 위치에 맞게 값들이 할당.
- 배열의 크기는 같을 필요는 없고, let이나 const 키워드로 사용 가능.
- 식별자 배열에 3개, 배열이 5개 있으면 앞에 3개만 할당.

- { 속성 이름, 속성 이름 }
= 객체
- { 식별자=속성이름, 식
별자=속성이름 } = 객체

속성 이름 그대로 꺼내서 출력하기

혼자 공부하는 파이썬 18000

다른 이름으로 속성 꺼내서 출력하기

혼자 공부하는 파이썬 18000

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 객체를 생성합니다.
7       const object = {
8         name: '혼자 공부하는 파이썬',
9         price: 18000,
10        publisher: '한빛미디어'
11      }
12
13     // 객체에서 변수를 추출합니다.
14     const { name, price } = object
15     console.log('# 속성 이름 그대로 꺼내서 출력하기')
16     console.log(name, price)
17     console.log('')
18
19     const { a=name, b=price } = object
20     console.log('# 다른 이름으로 속성 꺼내서 출력하기')
21     console.log(a, b)
22   </script>
23   </head>
24   <body></body>
25 </html>
```

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 사야 하는 물건 목록
7       const 물건_200301 = ['우유', '식빵']
8       const 물건_200302 = 물건_200301
9       물건_200302.push('고구마')
10      물건_200302.push('토마토')
11
12      // 출력
13      console.log(물건_200301)
14      console.log(물건_200302)
15    </script>
16  </head>
17  <body></body>
18 </html>

```

▼ Array(4) i
 0: "우유"
 1: "식빵"
 2: "고구마"
 3: "토마토"
 length: 4
 ▶ [[Prototype]]: Array(0)
 ▼ Array(4) i
 0: "우유"
 1: "식빵"
 2: "고구마"
 3: "토마토"
 length: 4
 ▶ [[Prototype]]: Array(0)

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       // 사야 하는 물건 목록
7       const 물건_200301 = ['우유', '식빵']
8       const 물건_200302 = [...물건_200301]
9       물건_200302.push('고구마')
10      물건_200302.push('토마토')
11
12      // 출력
13      console.log(물건_200301)
14      console.log(물건_200302)
15    </script>
16  </head>
17  <body></body>
18 </html>

```

▼ Array(2) i
 0: "우유"
 1: "식빵"
 length: 2
 ▶ [[Prototype]]: Array(0)
 ▼ Array(4) i
 0: "우유"
 1: "식빵"
 2: "고구마"
 3: "토마토"
 length: 4
 ▶ [[Prototype]]: Array(0)

chapter 7

D O M

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. The background is blurred.

DOM조작

- HTML페이지에 tag는 모두 element(요소)라고 함.
- 자바스크립트에서는 이를 DOM이라고 함.
- 결론은 같다.
- HTML 페이지는 코드를 위에서 아래로 순차적으로 실행함.
- head 태그내의 javascript 코드는 페이지가 다 로드되기 전에 body의 문서에 접근할 수 없음.
- 그래서 아래와 같은 접근이 필요

```
document.addEventListener('DOMContentLoaded', () => {
  const h1 = (text) => `<h1>${text}</h1>`
  document.body.innerHTML += h1('DOMContentLoaded 이벤트 발생')
})
```

- document.querySelector(선택자)
- document.querySelectorAll(선택자)
- document.head
- document.body
- document.title

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  document.addEventListener('DOMContentLoaded', () => {
7  |  |  |  // 요소를 읽어들입니다.,
8  |  |  |  const header = document.querySelector('h1')
9  |
10 |  |  |  // 텍스트와 스타일을 변경합니다.
11 |  |  |  header.textContent = 'HEADERS'
12 |  |  |  header.style.color = 'white'
13 |  |  |  header.style.backgroundColor = 'black'
14 |  |  |  header.style.padding = '10px'
15 |  |  |
16 |  |  |  }
17 |  |  </script>
18 |  |  </head>
19 |  |  <body>
20 |  |  |  <h1></h1>
21 |  |  </body>
22 |  </html>
```

- 문서객체.textContent : 입력된 문자열을 그대로 넣음.
- 문서객체.innerHTML : 입력된 문자열을 HTML 형식으로 넣음.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6              document.addEventListener('DOMContentLoaded', () => {
7                  const a = document.querySelector('#a')
8                  const b = document.querySelector('#b')
9
10                 a.textContent = '<h1>textContent 속성</h1>'
11                 b.innerHTML = '<h1>innerHTML 속성</h1>'
12             })
13         </script>
14     </head>
15     <body>
16         <div id="a"></div>
17         <div id="b"></div>
18     </body>
19 </html>
```

<h1>textContent 속성</h1>

innerHTML 속성

- 문서객체.setAttribute(속성 이름, 값) :특성 속성에 값을 지정
- 문서객체.getAttribute(속성 이름) : 특정 속성을 추출

```
1  <!DOCTYPE html>
2  <html>
3  |   <head>
4  |       <title></title>
5  |       <script>
6  |           document.addEventListener('DOMContentLoaded', () => {
7  |               const rects = document.querySelectorAll('.rect')
8  |
9  |               rects.forEach((rect, index) => {
10 |                   const width = (index + 1) * 100
11 |                   const src = `http://placekitten.com/${width}/250`
12 |                   rect.setAttribute('src', src)
13 |               })
14 |           })
15 |       </script>
16 |   </head>
17 |   <body>
18 |       <img class="rect">
19 |       <img class="rect">
20 |       <img class="rect">
21 |       <img class="rect">
22 |   </body>
23 | </html>
```

- div.style.backgroundColor
or
- CSS 활용과 유사.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       document.addEventListener('DOMContentLoaded', () => {
7         const divs = document.querySelectorAll('body > div')
8
9         divs.forEach((div, index) => {
10           console.log(div, index)
11           const val = index * 10
12           div.style.height = `10px`
13           div.style.backgroundColor = `rgba(${val}, ${val}, ${val})`
14         })
15       })
16     </script>
17   </head>
18   <body>
19     <!-- div 태그 25개 -->
20     <div></div><div></div><div></div><div></div><div></div><div></div><div></div>
21     <div></div><div></div><div></div><div></div><div></div><div></div><div></div>
22     <div></div><div></div><div></div><div></div><div></div><div></div><div></div>
23     <div></div><div></div><div></div><div></div><div></div><div></div><div></div>
24     <div></div><div></div><div></div><div></div><div></div><div></div><div></div>
25     <div></div><div></div><div></div><div></div><div></div><div></div><div></div>
26   </body>
27 </html>
```

- document.createElement(문서 객체 이름)
- 문서는 만들면 배치 되는게 아님.
- 부모(parent) --- 나 --- 자식(child) 의 구조에 appendChild(자식객체) 형태로 추가 해주어야 함.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        document.addEventListener('DOMContentLoaded', () => {
7          // 문서 객체 생성하기
8          const header = document.createElement('h1')
9
10         // 생성한 태그 조작하기
11         header.textContent = '문서 객체 동적으로 생성하기'
12         header.setAttribute('data-custom', '사용자 정의 속성')
13         header.style.color = 'white'
14         header.style.backgroundColor = 'black'
15
16         // h1 태그를 body 태그 아래에 추가하기
17         document.body.appendChild(header)
18       })
19     </script>
20   </head>
21   <body>
22
23   </body>
24 </html>
```

- 문서 객체의 부모(parent)는 언제나 하나.
- 문서 객체를 다른 문서 객체에 추가하면 문서 객체가 이동함.
- appendChild() 메소드는 문서객체 이동에도 사용.

- 조심히
- 부모객체.removeChild(자식객체)
 - 문서객체.parentNode.removeChild(문서 객체)

부모객체선택 삭제

삭제 관리

출처: 문자 공부하는 자바스크립트

- 모든 문서객체는 생성되거나 클릭되거나 마우스를 위로 올리거나 할 때 이벤트라는 것이 항상 발생.
- 그 이벤트가 발생할 때 실행할 함수는 `addEventListener()`
- 문서객체.addEventListener(이벤트 이름, 콜백 함수)
- 이때 콜백함수를 이벤트 리스터 또는 이벤트 핸들러 라고 함.
- 문서객체.removeEventListener(이벤트 이름, 이벤트 리스너)

클릭 횟수:2

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  document.addEventListener('DOMContentLoaded', () => {
7  |  |  |  |  let counter = 0
8  |  |  |  |  const h1 = document.querySelector('h1')
9  |
10 |  |  |  |  h1.addEventListener('click', (event) => {
11 |  |  |  |  |  counter++
12 |  |  |  |  |  h1.textContent = `클릭 횟수:${counter}`
13 |  |  |  |  })
14 |  |  |  })
15 |  |  |  </script>
16 |  |  |  <style>
17 |  |  |  |  h1{
18 |  |  |  |  |  /* 클릭을 여러 번 했을 때
19 |  |  |  |  |  글자가 선택되는 것을 막기 위한 스타일 */
20 |  |  |  |  user-select: none;
21 |  |  |  }
22 |  |  |  </style>
23 |  |  |  </head>
24 |  |  |  <body>
25 |  |  |  |  <h1>클릭 횟수: 0</h1>
26 |  |  |  </body>
27 |  </html>
```

chap 7

```
1  <!DOCTYPE html>
2  <html>
3  |<head>
4  |<title></title>
5  |<script>
6  |  document.addEventListener('DOMContentLoaded', () => {
7  |    let counter = 0
8  |    let isConnect = false
9
10 |    const h1 = document.querySelector('h1')
11 |    const p = document.querySelector('p')
12 |    const connectButton = document.querySelector('#connect')
13 |    const disconnectButton = document.querySelector('#disconnect')
14
15 |    const listener = (event) => {
16 |      h1.textContent = `클릭 횟수: ${counter++}`
17 |    }
18
19 |    connectButton.addEventListener('click', () => {
20 |      if (isConnect === false) {
21 |        h1.addEventListener('click', listener)
22 |        p.textContent = '이벤트 연결 상태: 연결'
23 |        isConnect = true
24 |      }
25 |    })
26 |  }
27 |</script>
28 |<style>
29 |  h1{
30 |    /* 클릭을 여러 번 했을 때
31 |     글자가 선택되는 것을 막기 위한 스타일 */
32 |    user-select: none;
33 |  }
34 |</style>
35 |</head>
36 |<body>
37 |  <h1>클릭 횟수: 0</h1>
38 |  <button id="connect">이벤트 연결</button>
39 |  <button id="disconnect">이벤트 제거</button>
40 |  <p>이벤트 연결 상태: 해제</p>
41 |</body>
42 |</html>
```

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. A large, semi-transparent white rectangular box is overlaid on the center-left portion of the image. Inside this box, the Korean text "이벤트 활용" is displayed in a bold, black, sans-serif font.

이벤트 활용

- 이벤트를 연결할 때, `addEventListener()` 메소드 사용.
- 표준 이벤트 모델
- 이전의 인라인 이벤트 모델은 하나의 리스너만 연결이 가능.
- 표준 이벤트 모델에서는 여러 개의 이벤트 리스너를 연결이 가능.

- keydown, keypress 이벤트는 웹브라우저에 따라 아시아권 문자를 제대로 처리 하지 못해서 일반적으로 keyup이벤트를 사용

이벤트	설명
keydown	키가 눌릴 때 실행. 꾹누르고 있을 때도, 입력될 때도 실행
keypress	키가 입력되었을 때 실행. 웹브라우저에 따라서 아시아권 문자를 제대로 처리하지 못하는 문제가 있음.
keyup	키보드에서 키가 떨어질 때 실행

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  document.addEventListener('DOMContentLoaded', () => {
7  |  |  |  |  const textarea = document.querySelector('textarea')
8  |  |  |  |  const h1 = document.querySelector('h1')
9
10 |  |  |  |  |  textarea.addEventListener('keyup', (event) => {
11 |  |  |  |  |  |  const length = textarea.value.length
12 |  |  |  |  |  |  h1.textContent = `글자 수: ${length}`
13 |  |  |  |  |  })
14 |  |  |  |  })
15 |  |  |  |  </script>
16 |  |  |  </head>
17 |  |  |  <body>
18 |  |  |  |  <h1></h1>
19 |  |  |  |  <textarea></textarea>
20 |  |  |  </body>
21 </html>
```

글자 수: 5

안녕하세요

- 키보드 이벤트가 발생할 때는 이벤트 객체로 어떤 키가 눌렸는지 관련 속성이 따라옴.

이벤트속성 이름	설명
Code	입력한 키
keyCode	입력한 키를 나타내는 숫자
altKey	
ctrlKey	
shiftKey	

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6      document.addEventListener('DOMContentLoaded', () => {
7          const h1 = document.querySelector('h1')
8          const print = (event) => {
9              let output = ''
10             output += `alt: ${event.altKey}<br>`
11             output += `ctrl: ${event.ctrlKey}<br>`
12             output += `shift: ${event.shiftKey}<br>`
13             output += `code: ${typeof(event.code) !== 'undefined' ? event.code : event.keyCode}<br>`
14             h1.innerHTML = output
15         }
16
17         document.addEventListener('keydown', print)
18         document.addEventListener('keyup', print)
19     })
20     </script>
21  </head>
22  <body>
23      <h1></h1>
24  </body>
25 </html>
```

alt: false
ctrl: false
shift: false
code: ShiftLeft

- 이벤트 리스너가 외부로 분리될 경우 블록외부에 있어서 접근이 안될수 있음. (오류 발생)
- 코드 규모가 커질 경우 외부로 분리하는경우가 다수 발생.
- 해결 방법
 - event.currentTarget 속성을 사용.
 - this키워드를 사용.

- 오류

```
<script>  
  const listener = (event) => {  
    const length = textarea.value.length → 현재 블록에서는 textarea 변수를  
    h1.textContent = `글자 수: ${length}` 사용할 수 없습니다.  
  }  
  
  document.addEventListener('DOMContentLoaded', () => { → 이벤트 리스너가 외부로  
    const textarea = document.querySelector('textarea') 분리되었습니다.  
    const h1 = document.querySelector('h1')  
    textarea.addEventListener('keyup', listener)  
  })  
</script>
```

- currentTarget으로 해결

```
<script>  
  const listener = (event) => {  
    const length = event.currentTarget.value.length → event.currentTarget가  
    h1.textContent = `글자 수: ${length}`  
  }  
  
  document.addEventListener('DOMContentLoaded', () => {  
    const textarea = document.querySelector('textarea')  
    const h1 = document.querySelector('h1')  
    textarea.addEventListener('keyup', listener)  
  })  
</script>
```

- 사용자로부터 어떠한 입력을 받을 때 사용하는 요소를 입력양식이라고 함. (input, textarea, button, select ...)
- 키보드 이벤트를 사용하면 입력을 했을 때, 실시간으로 변환도 가능.

dkdk

이메일 형식이 아닙니다: dkdk

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       document.addEventListener('DOMContentLoaded', () => {
7         const input = document.querySelector('input')
8         const p = document.querySelector('p')
9         const isEmail = (value) => {
10           // 골뱅이를 갖고 있고 && 골뱅이 뒤에 점이 있다면
11           return (value.indexOf('@') > 1)
12             && (value.split('@')[1].indexOf('.') > 1)
13         }
14
15         input.addEventListener('keyup', (event) => {
16           const value = event.currentTarget.value
17           if (isEmail(value)) {
18             p.style.color = 'green'
19             p.textContent = `이메일 형식입니다: ${value}`
20           } else {
21             p.style.color = 'red'
22             p.textContent = `이메일 형식이 아닙니다: ${value}`
23           }
24         })
25       }
26     </script>
27   </head>
28   <body>
29     <input type="text">
30     <p></p>
31   </body>
32 </html>
```

떡볶이 ▾

선택: 떡볶이

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       document.addEventListener('DOMContentLoaded', () => {
7         const select = document.querySelector('select')
8         const p = document.querySelector('p')
9
10        select.addEventListener('change', (event) => {
11          const options = event.currentTarget.options
12          const index = event.currentTarget.options.selectedIndex
13
14          p.textContent = `선택: ${options[index].textContent}`
15        })
16      })
17    </script>
18  </head>
19  <body>
20    <select>
21      <option>떡볶이</option>
22      <option>순대</option>
23      <option>오뎅</option>
24      <option>튀김</option>
25    </select>
26    <p>선택: 떡볶이</p>
27  </body>
28</html>
```



타이머 활성화

9초

```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  document.addEventListener('DOMContentLoaded', () => {
7  |  |  |  |  let [timer, timerId] = [0, 0]
8  |  |  |  |  const h1 = document.querySelector('h1')
9  |  |  |  |  const checkbox = document.querySelector('input')
10 |  |  |  |
11 |  |  |  |  checkbox.addEventListener('change', (event) => {
12 |  |  |  |  |  if (event.currentTarget.checked) {
13 |  |  |  |  |  |  // 체크 상태
14 |  |  |  |  |  |  timerId = setInterval(() => {
15 |  |  |  |  |  |  |  timer += 1
16 |  |  |  |  |  |  |  h1.textContent = `${timer}초`
17 |  |  |  |  |  |  |  }, 1000)
18 |  |  |  |  |  } else {
19 |  |  |  |  |  |  // 체크 해제 상태
20 |  |  |  |  |  |  clearInterval(timerId)
21 |  |  |  |  |  }
22 |  |  |  |  }
23 |  |  |  }
24 |  |  |  </script>
25 |  |  |  </head>
26 |  |  |  <body>
27 |  |  |  |  <input type="checkbox">
28 |  |  |  |  <span>타이머 활성화</span>
29 |  |  |  |  <h1></h1>
30 |  |  |  </body>
31 |  </html>
```

출처: 혼자 공부하는 자바스크립트

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       document.addEventListener('DOMContentLoaded', () => {
7         // 문서 객체 추출하기
8         const output = document.querySelector('#output')
9         const radios = document.querySelectorAll('[name=pet]')
10
11        // 모든 라디오 버튼에
12        radios.forEach((radio) => {
13          // 이벤트 연결
14          radio.addEventListener('change', (event) => {
15            const current = event.currentTarget
16            if (current.checked) {
17              output.textContent = `좋아하는 애완동물은 ${current.value}이시군요!`
18            }
19          })
20        })
21      })
22    </script>
23  </head>

```

좋아하는 애완동물을 선택해주세요

- 강아지 고양이 햄스터 기타
-

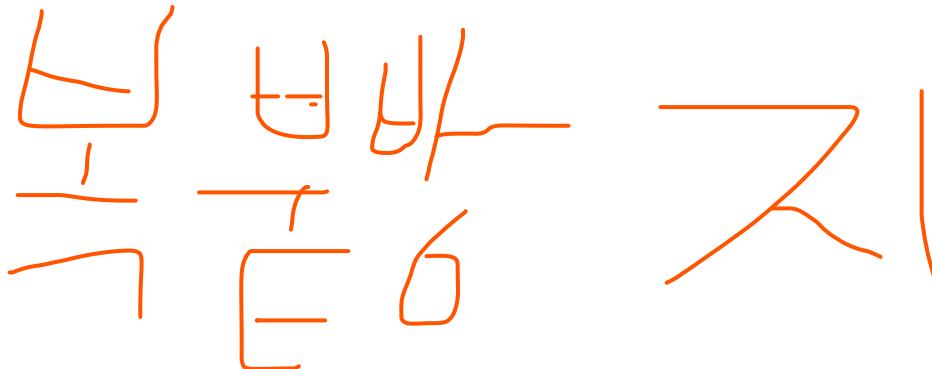
좋아하는 애완동물은 고양이이시군요!

```

24  <body>
25    <h3># 좋아하는 애완동물을 선택해주세요</h3>
26    <input type="radio" name="pet" value="강아지">
27    <span>강아지</span>
28    <input type="radio" name="pet" value="고양이">
29    <span>고양이</span>
30    <input type="radio" name="pet" value="햄스터">
31    <span>햄스터</span>
32    <input type="radio" name="pet" value="기타">
33    <span>기타</span>
34    <hr>
35    <h3 id="output"></h3>
36  </body>
37 </html>

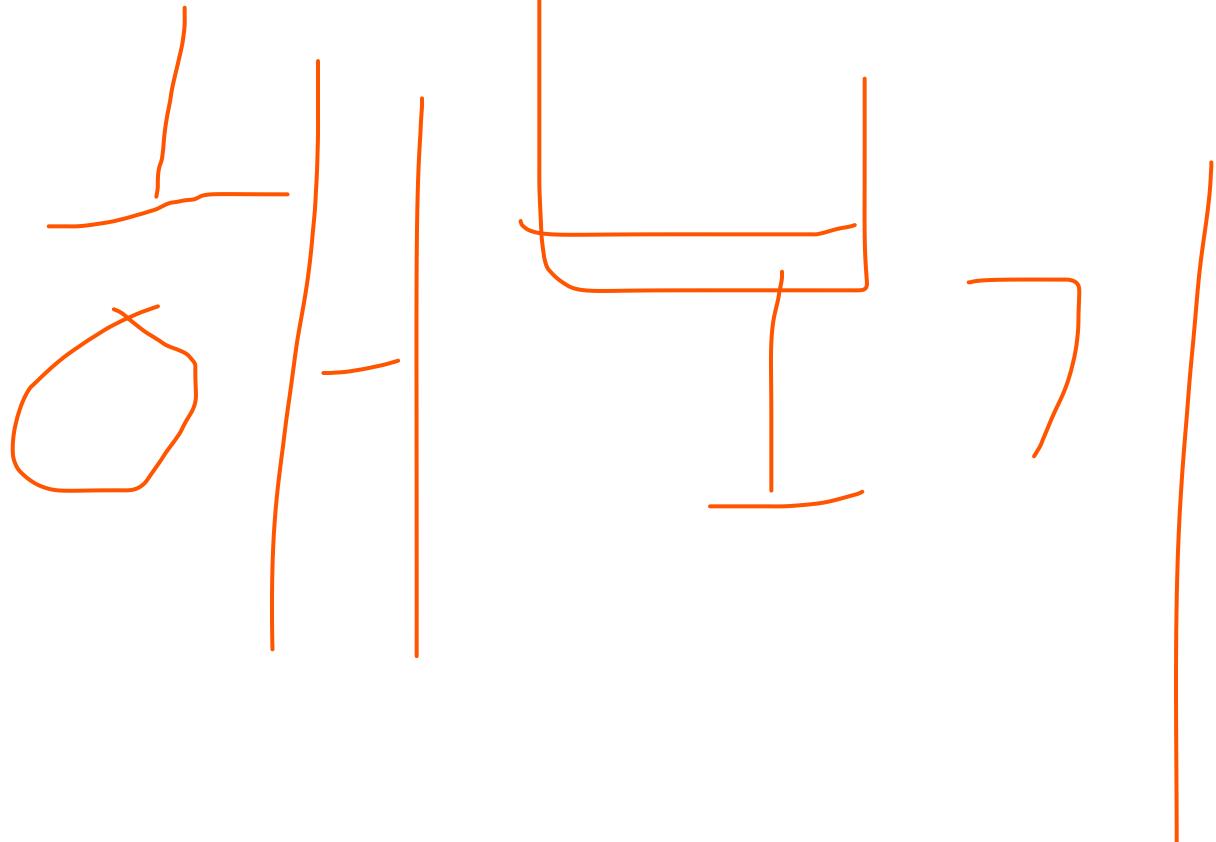
```

- 웹 브라우저에서 이미지에서 마우스 오른쪽 버튼을 클릭하면, Context Menu를 출력함.
- 어떤 이벤트가 발생했을 때 웹 브라우저가 기본적으로 처리해 주는 것을 기본 이벤트라고 함.
- 링크를 클릭했을 때 이동하는 것 / 제출 버튼을 눌렀을 때 이동하는 것 등등.
- 기본 이벤트를 제거할 때는 preventDefault() 메소드를 사용.



```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  |  <script>
6  |  |  |  document.addEventListener('DOMContentLoaded', () => {
7  |  |  |  |  const imgs = document.querySelectorAll('img')
8  |
9  |  |  |  |  imgs.forEach((img) => {
10 |  |  |  |  |  img.addEventListener('contextmenu', (event) => {
11 |  |  |  |  |  |  event.preventDefault()
12 |  |  |  |  |  })
13 |  |  |  |  })
14 |  |  |  |  })
15 |  |  |  |  </script>
16 |  |  |  </head>
17 |  |  <body>
18 |  |  |  
19 |  |  </body>
20 </html>
```

- 할 일 목록을 만들어 봅시다.
- 자바 스크립트로~~~
- 코드 분석해가며 완성.



- 오프라인에서 어플처럼 만들 때, 데이터 저장 공간.
- 메소드
 - localStorage.getItem(키) : 키에 저장된 값을 추출. 없으면 undefined
 - localStorage.setItem(키, 값) : 키에 값을 저장.
 - localStorage.removeItem(키) : 특정 키의 값을 제거
 - localStorage.clear() : 저장된 모든 값을 제거.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title></title>
5  <script>
6  document.addEventListener('DOMContentLoaded', () => {
7  const p = document.querySelector('p')
8  const input = document.querySelector('input')
9  const button = document.querySelector('button')
10
11  const savedValue = localStorage.getItem('input')
// localStorage.input도 가능합니다.
13  if (savedValue) {
14  input.value = savedValue
15  p.textContent = `이전 실행 때의 마지막 값: ${savedValue}`
16  }
17
18  input.addEventListener('keyup', (event) => {
19  const value = event.currentTarget.value
20  localStorage.setItem('input', value)
// localStorage.input = value도 가능합니다.
22  })

```

```

23
24  button.addEventListener('click', (event) => {
25    localStorage.clear()
26    input.value = ''
27  })
28  }
29  </script>
30  </head>
31  <body>
32  <p></p>
33  <button>지우기</button>
34  <input type="text">
35  </body>
36  </html>

```

이전 실행 때의 마지막 값: ddddddd

지우기

ddddd

chapter 8

예외처리

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. The background is blurred.

구문오류와 예외

- 구문 오류 : 프로그램 실행 전에 발생하는 오류
- 예외/런타임 오류 : 프로그램 실행 중에 발생하는 오류

- 다양한 오류
 - 괄호의 짹을 맞추지 않음.
 - 문자열을 열었는데, 닫지 않음.
- 구문오류가 있으면 웹 브라우저에서 코드 실행을 못함.

> `console.log("괄호를 닫지 않는 실수를 했습니다."`

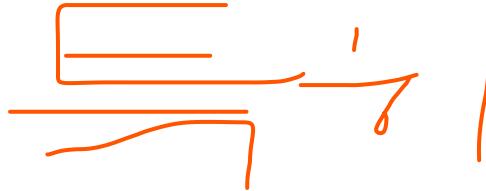
✖ `Uncaught SyntaxError: missing) after argument list` VM1806:1

- 자바 스크립트에서는 SyntaxError라고 출력되는 오류 이외의 모든 오류(TypeError, ReferenceError, RangeError)가 예외임.
- 실행중에 나면 다 예외임.
- 자바스크립트는 다른 프로그래밍 언어와 비교해서 굉장히 유연하기 때문에 예외를 발생할 가능성이 적은편.
- 배열길이 넘어서도 undefined 만 출력.
- 이런 유연함은 프로그램이 문제가 발생해도 죽지 않고 실행되면서 문제를 찾기 어려워지게 할 수 있음.

- 조건문을 사용해서 예외가 발생하지 않도록 만드는 것.

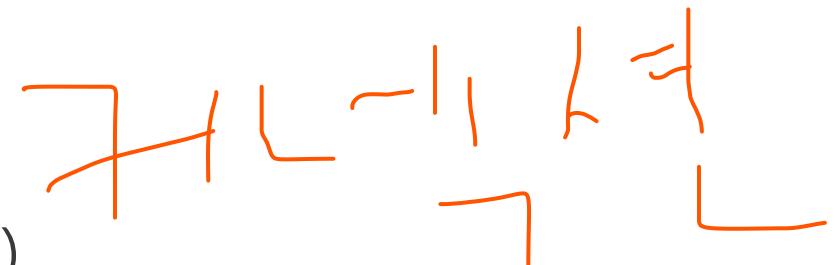
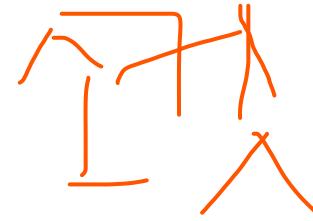
```
1  <!DOCTYPE html>
2  <html>
3  |  <head>
4  |  |  <title></title>
5  |  </head>
6  |  <body>
7
8  |  </body>
9  <script>
10 |  document.addEventListener('DOMContentLoaded', () => {
11 |  |  const h1 = document.querySelector('h1')
12 |  |  if (h1) {
13 |  |  |  h1.textContent = '안녕하세요'
14 |  |  } else {
15 |  |  |  console.log('h1 태그를 추출할 수 없습니다.')
16 |  |  }
17 |  })
18 </script>
19 </html>
```

- try catch finally 구문으로 처리.



- 기본적인 형태

```
try {  
    // 예외가 발생할 가능성이 있는 코드  
} catch (exception) {  
    // 예외가 발생 했을 때 실행할 코드  
} finally {  
    // 무조건 실행하 코드 (필요한 경우만 사용)  
}
```



```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title></title>
5          <script>
6              try {
7                  willExcept.byeBye()
8                  console.log("try 구문의 마지막 줄")
9              } catch (exception) {
10                  console.log("catch 구문의 마지막 줄")
11              } finally {
12                  console.log("finally 구문의 마지막 줄")
13              }
14          </script>
15      </head>
16      <body>
17          </body>
18      </html>
```

catch 구문의 마지막 줄

finally 구문의 마지막 줄

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. A large, semi-transparent white rectangular box is overlaid on the center-left portion of the image, containing the Korean text.

예외처리고급

- try catch 구문을 사용할 때, catch의 괄호 안에 입력하는 식별자 가 예외객체
- 아무 식별자나 입력해도 괜찮지만, 일반적으로 e 나 exception 이라는 식별자를 사용.
- 예외객체의 속성
 - name : 예외 이름
 - message : 예외 메시지

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title></title>
5  <script>
6      try {
7          const array = new Array(9999999999999999)
8      } catch (exception) {
9          console.log(exception)
10         console.log()
11         console.log(`예외 이름: ${exception.name}`)
12         console.log(`예외 메시지: ${exception.message}`)
13     }
14     </script>
15 </head>
16 <body>
17 </body>
18 </html>
```

RangeError: Invalid array length
at 8-2-1.html:7:23

예외 이름: RangeError

예외 메시지: Invalid array length

- 상황에 따라서 예외를 강제로 발생시켜야 하는 경우가 있음
- 예외를 강제로 발생시킬 때는 throw 키워드를 사용.
- 의도한 대로의 코드 사용을 유도할 때, 강제 발생 시킴.
- 이 exception을 try catch로 로직컬하게 처리를 할 수 있음.
- 웬만해서 오류를 안내기 때문에 일부러 예외를 발생시키는게 안전할 수 있음.
- 기본적인 형태
throw 문자열

throw new Error(문자열)

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title></title>
5  <script>
6  function test(object) {
7    if (object.a !== undefined && object.b !== undefined) {
8      console.log(object.a + object.b)
9    } else {
10      throw new Error("a 속성과 b 속성을 지정하지 않았습니다.")
11    }
12  }
13
14  test({})
15 </script>
16 </head>
17 <body>
18 </body>
19 </html>
```

✖ Uncaught Error: a 속성과 b 속성을 지정하지 않았습니다. [8-2-4.html:10](#)
at test ([8-2-4.html:10:17](#))
at [8-2-4.html:14:7](#)

chapter 9

클래스

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. The background is blurred.

기본기능

- OOP
 - 객체 지향 프로그래밍
 - 객체를 기반으로 설계하고 프로그래밍.
 - 클래스는 객체를 만들어내는 빵판.
 - 객체는 속성과 메소드를 가짐.
- OOP-추상화
 - 프로그램에서는 프로그램이 필요한 요소들 사용해서 객체로 표현할 필요가 있음
 - 이러한 과정을 추상화라고 함.
 - 포괄적인 개념만 정의하고 구체적인 건 하위에 구현을 통해서!!!

- 객체를 만드는 부분

- 객체를 활용하는 부분

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title></title>
5      <script>
6        // 객체를 선언합니다.
7        const students = []
8        students.push({ 이름: '구름', 국어: 87, 영어: 98, 수학: 88, 과학: 90 })
9        students.push({ 이름: '별이', 국어: 92, 영어: 98, 수학: 96, 과학: 88 })
10       students.push({ 이름: '겨울', 국어: 76, 영어: 96, 수학: 94, 과학: 86 })
11       students.push({ 이름: '바다', 국어: 98, 영어: 52, 수학: 98, 과학: 92 })

12       // 객체를 처리하는 함수를 선언합니다.
13       function getSumOf (student) {
14         return student.국어 + student.영어 + student.수학 + student.과학
15       }
16
17       function getAverageOf (student) {
18         return getSumOf(student) / 4
19       }
20
21       // 출력합니다.
22       let output = '이름\t총점\t평균\n'
23       for (const s of students) {
24         output += `${s.이름}\t${getSumOf(s)}점\t${getAverageOf(s)}점\n`
25       }
26       console.log(output)
27     </script>
28   </head>
29   <body>
30     </body>
31   </html>
32 
```

- 함수를 이용해서 객체를 생성.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <title></title>
5  <script>
6  function createStudent(이름, 국어, 영어, 수학, 과학) {
7  return {
8    // 속성을 선언합니다.
9    이름: 이름,
10   국어: 국어,
11   영어: 영어,
12   수학: 수학,
13   과학: 과학,
14   // 메소드를 선언합니다.
15   getSum () {
16     return this.국어 + this.영어 + this.수학 + this.과학
17   },
18   getAverage () {
19     return this.getSum() / 4
20   },
21   toString () {
22     return `${this.이름}\t${this.getSum()}점\t${this.getAverage()}점\n`
23   }
24 }
25
26

```

```

27   // 객체를 선언합니다.
28   const students = []
29   students.push(createStudent('구름', 87, 98, 88, 90))
30   students.push(createStudent('별이', 92, 98, 96, 88))
31   students.push(createStudent('겨울', 76, 96, 94, 86))
32   students.push(createStudent('바다', 98, 52, 98, 92))

33
34   // 출력합니다.
35   let output = '이름\t총점\t평균\n'
36   for (const s of students) {
37     output += s.toString()
38   }
39   console.log(output)
40   </script>
41   </head>
42   <body>
43   </body>
44   </html>

```

- 객체 하나 만들 때와의 비교
 - 코드의 양 감소.
 - 오탈자의 위험 감소(ㅠㅠ)
 - 마지막으로 속성과 메소드를 한 함수 내부에서 관리할 수 있어서 객체를 더 쉽게 유지보수 할 수 있음.
- 객체별로 getSum(), getAverage(), toString() 메소드를 생성.

- 선언
 - class 클래스 이름 { }
 - 클래스 이름은 첫글자는 대문자.(개발자들의 일반적 약속)
- 인스턴스(객체)
 - 클래스를 기반으로 만든 객체를 인스턴스라고 함. 객체라고도 함.
 - new 클래스 이름 형태로 생성.
- 클래스와 인스턴스의 차이
 - 클래스 : 이전에 살펴보았던 객체를 만드는 함수와 비슷한 것.(빵판)
 - 인스턴스(객체) : 이전에 만들었던 객체를 만드는 함수로 만든 객체와 비슷한 것. (붕어빵)

- 객체가 생성될 때 호출되는 함수
- 형식
 - class 클래스 이름 {
 - constructor (){
 - /* 생성자 코드 */ 
 - }
 - }
 - 인스턴스를 처음 생성할 때, 객체의 초기화를 위한 처리를 담당.

```
class Student {  
    constructor (이름, 국어, 영어, 수학, 과학) {  
        this.이름 = 이름  
        this.국어 = 국어  
        this.영어 = 영어  
        this.수학 = 수학  
        this.과학 = 과학  
    }  
}
```

- 클래스 내부에 선언



```
class Student {  
    constructor (이름, 국어, 영어, 수학, 과학) {  
        this.이름 = 이름  
        this.국어 = 국어  
        this.영어 = 영어  
        this.수학 = 수학  
        this.과학 = 과학  
    }  
  
    getSum () {  
        return this.국어 + this.영어 + this.수학 + this.과학  
    }  
    getAverage () {  
        return this.getSum() / 4  
    }  
    toString () {  
        return `${this.이름}\t${this.getSum()}점\t${this.getAverage()}점\n`  
    }  
}
```

A close-up photograph of a person's hands holding a white smartphone. The person is wearing a white button-down shirt. A large, semi-transparent white rectangular box is overlaid on the center-left portion of the image. Inside this box, the Korean word "고급기능" (Advanced Features) is written in a bold, black, sans-serif font.

고급기능

- Rectangle 클래스에 두개 메소드를 선언
- getPerimeter()
- getArea()
- 내부적으로 width, height 값을 가짐.



```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title></title>
5     <script>
6       class Rectangle {
7         constructor (width, height) {
8           this.width = width
9           this.height = height
10        }
11
12        // 사각형의 둘레를 구하는 메소드
13        getPerimeter () {
14          return 2 * (this.width + this.height)
15        }
16
17        // 사각형의 넓이를 구하는 메소드
18        getArea () {
19          return this.width * this.height
20        }
21
22
23      const rectangle = new Rectangle(10, 20)
24      console.log(`사각형의 둘레: ${rectangle.getPerimeter()}`)
25      console.log(`사각형의 넓이: ${rectangle.getArea()}`)
26    </script>
27  </head>
28  <body>
29  </body>
30 </html>

```

- 코드 중복이 많고 클래스간 차이 별로 없음.

```
class Rectangle {  
    constructor (width, height) {  
        this.width = width  
        this.height = height  
    }  
  
    // 사각형의 둘레를 구하는 메소드  
    getPerimeter () {  
        return 2 * (this.width + this.height)  
    }  
  
    // 사각형의 넓이를 구하는 메소드  
    getArea () {  
        return this.width * this.height  
    }  
}
```

```
class Square {  
    constructor (length) {  
        this.length = length  
    }  
  
    // 정사각형의 둘레를 구하는 메소드  
    getPerimeter () {  
        return 4 * this.length  
    }  
  
    // 정사각형의 넓이를 구하는 메소드  
    getArea () {  
        return this.length * this.length  
    }  
}
```

- 위와 같은 경우, 상속으로 정의
- 선언
 - class 클래스 이름 extends 부모클래스 이름 { }
- 특징
 - 부모 클래스가 가지고 있는 속성과 메소드를 물려받음.
 - 자식 클래스에서 super() 함수는 부모의 생성자를 나타내는 함수.

- 상속을 통해서 코드 간결화.

```
class Square {  
    constructor (length) {  
        this.length = length  
    }  
  
    // 정사각형의 둘레를 구하는 메소드  
    getPerimeter () {  
        return 4 * this.length  
    }  
  
    // 정사각형의 넓이를 구하는 메소드  
    getArea () {  
        return this.length * this.length  
    }  
}
```



```
// 정사각형 클래스  
class Square extends Rectangle {  
    constructor (length) {  
        super(length, length)  
    }  
}
```

- 클래스 사용자가 클래스 속성(또는 메소드)을 의도하지 않은 방향으로 사용하는 것을 막아 클래스의 안정성을 확보하기 위해 나온 문법이 private 속성과 메소드.
- 형식
 - class 클래스 이름 {
 - #속성 이름
 - #메소드 이름(){
 - }
 - }
- 속성과 메소드 이름 앞에 #
- private 속성은 사용하기 전에 미리 외부에 어떤 속성을 private 속성으로 사용할지 선언이 필요.

- length라는 걸 private으로 사용하기 전에 생성자 위에 #length로 미리 선언.
- 이럴 경우 클래스 외부에서는 접근이 불가.
- 강제 시도 시, Uncaught SyntaxError 발생.

```
<script>
  // 사각형 클래스
  class Square {
    #length

    constructor (length) {
      if (length <= 0) {
        throw '길이는 0보다 커야 합니다.'
      }
      this.#length = length
    }

    getPerimeter () { return 4 * this.#length }
    getArea () { return this.#length * this.#length }
  }

  // 클래스 사용하기
  const square = new Square(10)
  console.log(`정사각형의 둘레: ${square.getPerimeter()}`)
  console.log(`정사각형의 넓이: ${square.getArea()}`)
</script>
```

- private로 선언된 속성에는 클래스 외부에서 접근이 불가.
- 따라서 속성을 변경하기 위해서는 속성을 읽고 쓸 수 있는 메소드가 필요.
- `getXXX()` 메소드처럼 속성 값을 확인할 때 사용하는 메소드를 **게터**, `setXXX()` 메소드처럼 속성에 값을 지정할 때 사용하는 메소드를 **세터**라고 함.(getter / setter)
- 형식
 - class 클래스 이름 {
 - get 이름 () { return 값 }
 - set 이름 (value) { }
 - }

|

- set / get 뒤에 띄어 쓰고 메소드 이름.
- 클래스를 활용하는 쪽에서는 단순하게 속성을 사용하는 형태처럼 게터와 세터 사용이 가능.

```

class Square {
  #length

  constructor (length) {
    this.length = length
  }

  get length () {
    return this.#length
  }

  get perimeter () {
    return this.#length * 4
  }

  get area () {
    return this.#length * this.#length
  }

  set length (length) {
    if (length <= 0) {
      throw '길이는 0보다 커야 합니다.'
    }
    this.#length = length
  }
}

```

- static 속성과 메소드는 인스턴스를 만들지 않고 사용할 수 있는 속성과 메소드.
- 형식
 - class 클래스 이름 {
 - static 속성 = 값
 - static 메소드 () {
 - }
 - }
 - 사용
 - 클래스 이름.속성
 - 클래스 이름.메소드()

chap 9

static 속성과 메소드

```
class Square {  
    #length  
  
    static #conuter = 0  
    static get counter () {  
        return Square.#conuter  
    }  
  
    constructor (length) {  
        this.length = length  
        Square.#conuter += 1  
    }  
  
    static perimeterOf (length) {  
        return length * 4  
    }  
    static areaOf (length) {  
        return length * length  
    }  
  
    get length () { return this.#length }  
    get perimeter () { return this.#length * 4 }  
    get area () { return this.#length * this.#length }  
  
    set length (length) {  
        if (length < 10) {  
            throw '길이는 0보다 커야 합니다.'  
        }  
        this.#length = length  
    }  
}
```

Static 속성과 메소드

Square 가 가지고 있는

// static 속성 사용하기
const squareA = new Square(10)
const squareB = new Square(20)
const squareC = new Square(30)
console.log(`지금까지 생성된 Square 인스턴스는 \${Square.counter}개입니다.`)
// static 메소드 사용하기
console.log(`한 변의 길이가 20인 정사각형의 둘레는 \${Square.perimeterOf(20)}입니다.`)
console.log(`한 변의 길이가 30인 정사각형의 둘레는 \${Square.areaOf(30)}입니다.`)

- 부모가 갖고 있는 함수를 자식에서 다시 선언해서 덮어쓰는 것.
- a()를 child에서 재정의
- 부모의 a()를 사용 할 때는 super.a()

```
// 클래스를 선언합니다.
class LifeCycle {
  call () {
    this.a()
    this.b()
    this.c()
  }

  a () { console.log('a() 메소드를 호출합니다.') }
  b () { console.log('b() 메소드를 호출합니다.') }
  c () { console.log('c() 메소드를 호출합니다.') }
}

class Child extends LifeCycle {
  a () {
    console.log('자식의 a() 메소드입니다.')
  }
}

// 인스턴스를 생성합니다.
new Child().call()
```

chapter 10

기타

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. The background is blurred.

사이트제작

이번 강

실습

HTML + CSS + javascript

<https://www.yes24.com/Product/Goods/97405662>

특별 부록
PDF 전자책

세상의 속도를
따라잡고 싶다면

Do it!



한 권으로 끝내는 웹 기본 교과서

HTML+CSS+ 자바스크립트 웹 표준의 정석

코딩 왕초보도 OK! 기초부터 활용까지 완·전·정·복

웹사이트 만들기

웹 분야 베스트셀러 저자! 고경화 지음

이지스 퍼블리싱

A close-up photograph of a person's hands holding a silver smartphone. The person is wearing a white button-down shirt. The background is blurred.

#KEYWORDS



감사합니다