# Javascript 기초

## 배열과 객체

김태민

저번에 우리는

# 반복문



반복문 loop

조건문과 더불어 JS의 핵심 문법

지정한 조건이 참(true)으로 평가되는 동안 지정한 블록문을 반복해서 실행하는 문법

while, do...while, for문

# 반복문



Break 문

종료 조건을 만족하지 않아도 인위적으로 반복문을 종료하는 방법
어떤 반복문이라도 break 문을 만나면 반복문 종료

반복문 때문에 0부터 9까지 출력해야 하지만, 변수 i가 5가 되는 순간 break 문이 실행되어 반복문이 종료됨

```
for(let i = 0; i < 10; i++){
    console.log(i);
    if(i === 5) break;
}
```

# 반복문



continue 문

break 문이 반복문을 즉시 종료하는 명령이라면, continue 문은 반복문을 건너뛰고 실행하라는 명령

반복문의 블록문에서 continue 문을 만나면 해당 반복 실행만 건너뜀

1부터 10까지 반복하는데, 홀수일 때는 반복문을 건너뛰고 짝수만 출력

```
for(let i = 1; i <= 10; i++){
  if(i % 2 === 1) continue;
  console.log(i);
}
```

# 자바스크립트 함수



함수

어떤 목적을 가지고 작성한 코드를 모아 둔 블록문

코드를 함수로 만들면 함수를 호출해 함수 내부에 모아 둔 여러 줄의 코드를 한 번에 실행 가능

구구단 3단을 계산하는 코드를 프로그램의 여러 군데에서 사용해야 한다.

```
for(let i = 1; i <= 9; i++){
    console.log(`3 * ${i} = ${3 * i}`);
}
```

# 자바스크립트 함수



함수정의 – 화살표 함수

ES6에서 추가된 함수 정의 방법

함수를 호출하려면 정의된 함수를 변수에 할당하는 방법인 함수 표현식을 사용해야 함

```
const gugudan = () => {
    for(let i = 1; i <= 9; i++){
        console.log(`3 * ${i} = ${3 * i}`);
    }
};
gugudan();
```

# 자바스크립트 함수



매개변수와 인수

3단이 아니라 5단, 8단처럼 다른 단을 출력하려면...

함수 내부의 블록문을 보면 빨간색 네모 안의 단수만 다르고 나머지는 동일함

```
function gugudan5(){
    for(let i=1; i<=9; i++){
        console.log(`5 * ${i} = ${5 * i}`);
    }
}
```

```
function gugudan8(){
    for(let i=1; i<=9; i++){
        console.log(`8 * ${i} = ${8 * i}`);
    }
}
```

# 실습 환경

# 실습환경



OS : Window / Mac

브라우저 : Chrome

에디터 : VS Code https://code.visualstudio.com/

VS Code 익스텐션 : ESLint, Prettier, HTML CSS Support, HTML to CSS autocompletion, Auto Rename Tag, Auto Close Tag, htmltagwrap

Git : git bash, github 가입

# 객체

객체

객체라는 용어의 범위는 자바스크립트에서 매우 포괄적
자료형의 관점에서 보면 키(key)와 값(value)으로 구성된 속성의 집합
객체는 {}를 이용해 생성할 수 있는데, 이런 방법을 리터럴(literal) 방식으로 객체를 생성했다고 표현

```
const person = {};
```

속성이 한 개도 없는 객체를 빈 객체라고 합니다.

사람에 대한 객체를 생성한다면 다음과 같은 속성을 지정해 객체를 생성할 수 있음

```
const person = {"Hong Gildong"};
```

객체

객체 안에 또 다른 객체나 함수가 들어갈 수도 있음

객체에서 속성의 값으로 함수가 들어갈 때는
보통 함수라고 하지 않고,
**메서드**(method) 라고 부름

```
const person = {
    name:{
        firstName:"Gildong",
        lastName:"Hong"
    },
    age:20,
    isAdult:true,
    printInfo:function(){
        console.log('printInfo');
    }
};
```

## 객체 속성 접근하기 – 대괄호 연산자 []

[]를 사용해 객체의 속성에 접근하는 방법, 배열에서도 사용할 수 있음
객체의 속성에 접근하려면 객체명 뒤에 대괄호를 붙이고 대괄호 안에 키를 입력
키는 반드시 큰따옴표나 작은따옴표로 감싼 문자열 형태로 작성해야 함

```
const person = {
    name:"Hong Gildong",
    age:20
};
console.log(person["name"]); // Hong GilDong
console.log(person["age"]);  // 20
```

## 객체 속성 접근하기 – 대괄호 연산자 []

따옴표를 생략하고 키를 작성하면 객체에서 키가 아닌 name이라는 변수를 찾게 되어 오류가 발생

```
const person = {
    name:"Hong Gildong"
};
console.log(person[name]); // ReferenceError: name is not defined
```

## 객체 속성 접근하기 – 대괄호 연산자 []

객체의 속성값이 배열이나 객체 리터럴, 함수라면 어떻게 접근해야 할까

```
const person = {
    name:{
        firstName:"Gildong",
        lastName:"Hong"
    },
    likes:["apple", "samsung"],
    printHello:function(){
        return "hello";
    }
};
```

## 객체 속성 접근하기 – 대괄호 연산자 []

객체의 속성값이 배열이나 객체 리터럴, 함수라면 어떻게 접근해야 할까

```
const person = {
    name:{
        firstName:"Gildong",
        lastName:"Hong"
    },
    likes:["apple", "samsung"],
    printHello:function(){
        return "hello";
    }
};
```

```
// { firstName:'Gildong', lastName:'Hong' }
console.log(person["name"]);

// person 객체의 name 속성에 값으로 할당된 객체의 firstName 속성에 접
console.log(person["name"]["firstName"]); // Gildong
```

## 객체 속성 접근하기 – 대괄호 연산자 []

객체의 속성값이 배열이나 객체 리터럴, 함수라면 어떻게 접근해야 할까

```
const person = {
    name:{
        firstName:"Gildong",
        lastName:"Hong"
    },
    likes:["apple", "samsung"],
    printHello:function(){
        return "hello";
    }
};
```

```
// [ 'apple', 'samsung' ]
console.log(person["likes"]);

console.log(person["likes"][0]); // apple
console.log(person["likes"][1]); // samsung
```

## 객체 속성 접근하기 – 대괄호 연산자 []

객체의 속성값이 배열이나 객체 리터럴, 함수라면 어떻게 접근해야 할까

```
const person = {
    name:{
        firstName:"Gildong",
        lastName:"Hong"
    },
    likes:["apple", "samsung"],
    printHello:function(){
        return "hello";
    }
};
```

```
// [Function: printHello]
console.log(person["printHello"]);

console.log(person["printHello"]()); // hello
```

# 객체



객체 속성 접근하기 – 마침표 연산자 .

. 을 사용해 객체의 속성에 접근하는 방법
배열에서도 사용할 수 있음

객체 속성에 접근할 때
키를 큰따옴표("")나 작은따옴표('')로
감싸면 오류가 발생

```
const person = {
    name:{
        firstName:"Gildong",
        lastName:"Hong"
    },
    age:20,
    likes:["apple", "samsung"],
    printHello:function(){
        return "hello";
    }
};
console.log(person.name.lastName); // Hong
console.log(person.age); // 20
console.log(person.likes[0]); // apple
console.log(person.printHello()); // hello
```

객체 속성 접근하기 – 마침표 연산자 .

객체 속성에 접근할 때 대괄호 연산자와 마침표 연산자 중 어느 방법을 사용하더라도 상관없음

하지만 일반적으로 마침표 연산자를 많이 사용함

그래서 공백이 있는 키를 지양함

```
const person = {
  "phone number":"010-000-0000"
};
console.log(person["phone number"]); // 010-000-0000
console.log(person.phone number); // error
```

## 객체 속성 값 변경하기

객체로 정의된 값을 바꾸고 싶다면 키로 속성에 접근해서 값을 재할당

변수 person에 할당된 객체의 속성에 키로 접근해 값을 변경

```
const person = {
    name:"Hong Gildong"
};
person.name = "Kim"; //  또는 person["name"] = "Kim";
console.log(person.name); // Kim
```

객체 속성 동적으로 추가하기

객체 속성에 키로 접근한 키가 객체에 없다면 즉,
객체에 없는 속성이라면 해당 키와 값으로 구성된 새로운 속성이 객체에 추가됨.

```
const person = {};
console.log(person); // {}
person.name = "Hong Gildong";
console.log(person); // { name: 'Hong Gildong' }
```

객체 리터럴 방식으로 빈 객체를 생성하고 변수에 할당
객체의 속성에 접근해 값을 변경할 때처럼 키에 값을 할당
person 객체를 출력해 보면 처음에는 빈 객체였는데, 나중에 name 속성이 추가된 것을 볼 수 있음

## 객체 속성 동적으로 추가하기

객체 식별자와 키에 마침표 연산자를 사용하면 객체의 속성에 접근

할당 연산자로 값을 할당하면 값이 변경되거나 새로운 속성이 추가

객체 속성의 값이 함수나 배열, 객체 리터럴일 때도 같은 방법으로 값을 변경하거나 새로운 속성을 추가 가능

```
const person = {};
    person.name = {
        firstName:"GilDong",
        lastName:"Hong"
    };
    person.likes = ["apple", "samsung"];
    person.printHello = function(){
        return "hello";
    }
```

객체 속성 동적으로 삭제하기

객체 속성에 접근할 때 앞에 delete 키워드를 명시하면 해당 속성 삭제

```
const person = {
  name:"Hong Gildong"
};
delete person.name; // 또는 delete person["name"]
console.log(person); // {} 출력
```

## 문자열 String 객체

String 객체는 기본 자료형에서 문자열을 다룸

문자열에서 사용할 수 있는 속성과 메서드가 정의되어 있음

실무에서 자주 사용하는 String 객체의 속성과 메서드들이 있음

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

## 문자열 String 객체

| 구분 | | 설명 |
|---|---|---|
| 속성 | length | 문자열의 길이를 반환합니다. |
| 메서드 | includes() | 메서드의 매개변수에 인자로 전달되는 문자열이 대상 문자열에 포함되어 있으면 true, 아니면 false를 반환합니다. |
| | replace() | 대상 문자열에서 메서드의 매개변수에 인자로 전달되는 문자열과 일치하는 한 부분을 찾아서 다른 데이터로 변경한 새로운 문자열을 반환합니다. |
| | replaceAll() | 대상 문자열에서 메서드의 매개변수에 인자로 전달되는 문자열과 일치하는 모든 부분을 찾아서 다른 데이터로 변경한 새로운 문자열을 반환합니다. |
| | split() | 메서드의 매개변수에 인자로 전달되는 구분자를 이용해 대상 문자열을 여러 개의 문자열로 분리하고, 분리한 문자열을 새로운 배열로 반환합니다. |
| | toUpperCase() | 대상 문자열을 대문자로 변경해 반환합니다. |
| | trim() | 대상 문자열의 앞뒤 공백을 제거한 값을 반환합니다. |
| | indexOf() | 대상 문자열과 일치하는 첫 번째 문자의 인덱스를 반환합니다. |

## 문자열 String 객체

```
const pw = "124";
if(pw.length < 4){
    console.log("비밀번호는 최소 4자리 이상 입력해 주세요.");
}
```

```
const email = "test!naver.com";
if(email.includes("@") === false){
    console.log("올바른 이메일 형식이 아닙니다.");
}
```

```
const email = "test!naver.com";
if(email.indexOf("@") ===> -1){
    console.log("올바른 이메일 형식이 아닙니다.");
}
```

다양한 객체

Date : 날짜 및 시간과 관련 있는 메서드 정의. 날짜/시간 계산 등에 사용 가능

Math : 수학 연산과 관련한 메서드 정의. 반올림/내림/올림/난수구하기 등

브라우저 객체 모델(BOM) : 웹 브라우저에서 제공하는 객체.

최상위에 window 객체

그 아래에 웹 문서와 관련 있는 기능이 모여 있는 document 객체,

현재 페이지의 URL 정보가 담겨 있는 location 객체,

방문 기록 정보가 담겨 있는 history 객체,

웹 브라우저의 정보가 담겨 있는 navigator 객체,

방문자의 화면 정보를 담고 있는 screen 객체

# 배열

# 배열



JavaScript 배열

배열(Array)은 여러 값을 순서대로 저장하는 데이터 구조

각 값은 인덱스(0부터 시작하는 숫자)로 접근

동적 크기: 필요에 따라 길이 조정 가능

다양한 데이터 타입 저장 가능(숫자, 문자열, 객체 등)

웹 개발에서 데이터 목록(예: 제품 목록, 사용자 리스트) 관리에 필수

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

## 배열 선언

### 리터럴 표기법( [] )과 Array 생성자( new Array() )로 선언

```
// 배열 선언
let fruits = ["사과", "바나나", "오렌지];  // let fruits = new Array("사과", "바나나", "오렌지"); 와 동일
console.log(fruits); // ["사과", "바나나", "오렌지"]
console.log(fruits[1]); // "바나나"
console.log(fruits.length); // 3
// 다양한 데이터 타입
let mixed = [1, "Hello", true, { name: "홍길동" }];
console.log(mixed); // [1, "Hello", true, { name: "홍길동" }]
```

## 배열 push

배열 끝에 요소 추가. 새로운 배열 길이 반환

```
let fruits = ["사과", "바나나"];
fruits.push("오렌지");
console.log(fruits); // ["사과", "바나나", "오렌지"]
console.log(fruits.push("망고")); // 4 (새 길이)
```

## 배열 pop

배열 끝에서 요소 제거, 제거된 요소 반환

```
let fruits = ["사과", "바나나", "오렌지"];
let removed = fruits.pop();
console.log(fruits); // ["사과", "바나나"]
console.log(removed); // "오렌지"
```

## 배열 shift

배열 처음에서 요소 제거, 제거된 요소 반환

```
let fruits = ["사과", "바나나", "오렌지"];
let removed = fruits.shift();
console.log(fruits); // ["바나나", "오렌지"]
console.log(removed); // "사과"
```

## 배열 unshift

배열 처음에 요소 추가. 새로운 배열 길이 반환

```
let fruits = ["사과", "바나나"];
fruits.unshift("망고");
console.log(fruits); // ["망고", "사과", "바나나"]
console.log(fruits.unshift("포도")); // 4 (새 길이)
```

## 배열 splice

특정 인덱스에서 요소 제거/추가/교체. 제거된 요소 배열 반환

array.splice(start, deleteCount, item1, item2, ...)

- start: 시작 인덱스
- deleteCount: 제거할 요소 수
- item1, item2, ...: 추가할 요소

```
let fruits = ["사과", "바나나", "오렌지"];
// 1번 인덱스에서 1개 제거, "망고" 추가
let removed = fruits.splice(1, 1, "망고");
console.log(fruits); // ["사과", "망고", "오렌지"]
console.log(removed); // ["바나나"]

// 1번 인덱스에 "포도" 추가 (제거 없음)
fruits.splice(1, 0, "포도");
console.log(fruits); // ["사과", "포도", "망고", "오렌지"]
```

배열 map

정의: 배열의 각 요소를 변환해 새로운 배열 반환
문법: array.map(callback(element, index, array))
특징: 원본 배열 변경 안 함

```
let numbers = [1, 2, 3];
let doubled = numbers.map(num => num * 2);
console.log(doubled); // [2, 4, 6]
console.log(numbers); // [1, 2, 3] (원본 유지)

// 객체 배열 변환
let users = [
    { name: "홍길동", age: 25 },
    { name: "김영희", age: 30 }
];
let names = users.map(user => user.name);
console.log(names); // ["홍길동", "김영희"]
```

배열 filter

정의: 조건을 만족하는 요소만 골라 새로운 배열 반환

문법: array.filter(callback(element, index, array))

특징: 원본 배열 변경 안 함

```
let numbers = [1, 2, 3, 4, 5];
let evens = numbers.filter(num => num % 2 === 0);
console.log(evens); // [2, 4]
console.log(numbers); // [1, 2, 3, 4, 5] (원본 유지)


// 객체 배열 필터링
let users = [
    { name: "홍길동", age: 25 },
    { name: "김영희", age: 30 },
    { name: "이철수", age: 20 }
];
let adults = users.filter(user => user.age >= 25);
console.log(adults); // [{ name: "홍길동", age: 25 },
{ name: "김영희", age: 30 }]
```

배열 reduce

정의: 배열을 순회하며 단일 값으로 축소

문법:

array.reduce(callback(accumulator, element, index, array), initialValue)

- accumulator: 누적된 결과
- initialValue: 초기값(선택)

특징: 합계, 문자열 결합, 복잡한 데이터 변환에 유용

## 배열 reduce

```
let numbers = [1, 2, 3, 4];
let sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum); // 10 (0 + 1 + 2 + 3 + 4)

// 객체 배열에서 총합 계산
let cart = [
    { item: "책", price: 10000 },
    { item: "펜", price: 2000 }
];
let totalPrice = cart.reduce((acc, product) => acc + product.price, 0);
console.log(totalPrice); // 12000
```

# 오늘 우리는

객체

객체라는 용어의 범위는 자바스크립트에서 매우 포괄적
자료형의 관점에서 보면 키(key)와 값(value)으로 구성된 속성의 집합
객체는 {}를 이용해 생성할 수 있는데, 이런 방법을 리터럴(literal) 방식으로 객체를 생성했다고 표현

```
const person = {};
```

속성이 한 개도 없는 객체를 빈 객체라고 합니다.

사람에 대한 객체를 생성한다면 다음과 같은 속성을 지정해 객체를 생성할 수 있음

```
const person = {"Hong Gildong"};
```

# 배열



JavaScript 배열

배열(Array)은 여러 값을 순서대로 저장하는 데이터 구조

각 값은 인덱스(0부터 시작하는 숫자)로 접근

동적 크기: 필요에 따라 길이 조정 가능

다양한 데이터 타입 저장 가능(숫자, 문자열, 객체 등)

웹 개발에서 데이터 목록(예: 제품 목록, 사용자 리스트) 관리에 필수

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

# 감사합니다