섹션 2. JavaScript 응용

# ing	10
# goal	10
Σ 수식	✅

▼ Truthy & Falsy

```
let a = "";

if(a){
  console.log("TRUE");
} else{
  console.log("FALSE");
}  // 빈 문자열을 조건문에 넣으면 거짓으로 인식
```

TRUTHY	“비어 있지 않은 문자열”	[ ] // 빈 배열	{ } // 객체 리터럴	숫자(Infinity 포함)
FALSY	undefined	null / NaN	0	“ “ // 빈 문자열

```
const getName = (person) =>{
  if(!person){  // undefined & null 모두 falsy라 조건문에 활용 가능
    return "객체가 아닙니다";
  }
  return person.name;
};

let person;
const name = getName(person);
console.log(name);
```

▼ 삼항 연산자

```
// 주어진 숫자가 양수인지 음수인지 판별하는 함수
let a = 3;

if(a>=0){
```

```
  console.log("양수");
} else{
  console.log("음수");
}  // 이 조건식을 한줄로 표현하는 게 삼항연산자

// 조건문?true:false
a>=0 ? console.log("양수") : console.log("음수");
```

```
// 주어진 배열이 비어 있는지 여부 확인하는 함수
let a = [];

if(a.length===0){
  console.log("빈 배열");
} else{
  console.log("안 빈 배열");
}

// 삼항연산자
a.length===0 ? console.log("빈 배열") : console.log("안 빈 배열");

---

// 값을 리턴
let a = [1, 23];

const arrayStatus = a.length===0 ? "빈 배열" : "안 빈 배열";
console.log(arrayStatus);
```

```
// trusy와 falsy 이용하기
// 주어진 값이 null이거나 undefined가 아닌지 판단하는 함수
let a;

const result = a ? true : false;
console.log(result);
```

```
// 삼항연산자의 중복 사용
// TODO: 학점 계산 프로그램
// 90점 이상 A+, 50점 이상 B+, 둘 다 아니면 F
let score=100;

// 참이면 A+ 출력, 아니면 한 번 더 비교
score>=90 ? console.log("A+") : score>=50 ? : console.log("B+") : console.log("F");

// if 조건문으로 변경하기
if(score>=90){
  console.log("A+");
} else if(score>=50){
  console.log("B+");
} else{
  console.log("F");
}
```

▼ 단락회로 평가

```
// 논리연산자의 연산 순서를 이용(오른쪽 -> 왼쪽)
const getName = (person) => {
  if(!person){
    return "객체가 아닙니다";
  }
  return person.name;
};

---

const getName = (person) => {
  return person && person.name;
};  // person이 falsy이기 때문에 AND 연산자에서 뒤에 오는 값 고려할 필요 X

let person;
const name = getName(person);
console.log(name);
```

```
const getName = (person) => {
  const name = person && person.name;  // 둘 다 trusy -> name 반환
  return name || "객체가 아닙니다";  // name이 trusy라 이름 반환
};

let person = {name: "조은정"};
const name = getName(person);
console.log(name);  // 조은정
```

▼ 조건문 Upgrade

```
// 전달받은 음식이 한식인지 확인하는 코드
function isKoreanFood(food){
  if(food==="불고기" || food==="비빔밥" || food==="떡볶이"){
    return true;
  }
  return false;
}

---

function isKoreanFood(food){
  if(["불고기", "떡볶이", "비빔밥"].includes(food)){
    return true;
  }
  return false;
}

const food1 = isKoreanFood("불고기");  // true
const food2 = isKoreanFood("파스타");  // false
```

```
// 주어진 값에 따라 각각 다른 결과물을 반환하는 함수
const getMeal = (mealType)=>{
  if(mealType === "한식") return "불고기";
  if(mealType === "양식") return "파스타";
  if(mealType === "중식") return "멘보샤";
  if(mealType === "일식") return "초밥";
```

```
  return "굶기";
};

---

const meal = {
  한식: "불고기",
  중식: "멘보샤",
  일식: "초밥",
  양식: "스테이크",
  인도식: "카레"
};

const getMeal = (mealType) => {
  return meal[mealType] || "굶기";
};

console.log(getMeal("한식"));  // 불고기
console.log(getMeal("중식"));  // 멘보샤
console.log(getMeal());  // 굶기
```

▼ 비 구조화 할당

```
let arr = ["one", "two", "three"];

let one = arr[0];
let two = arr[1];
let three = arr[2];

console.log(one, two, three);  // one, two, three
```

```
let arr = ["one", "two", "three"];

// 배열의 기본 변수 비 구조화 할당
let [one, two, three] = arr;

---

// 배열의 선언 분리 비 구조화 할당
let [one, two, three, four="four"] = ["one", "two", "three"];
```

```
// 비 구조화 할당을 swap에 활용 가능
let a=10;
let b=20;
let tmp=0;

// tmp=a; a=b; b=tmp;

[a, b] = [b, a];
```

```
let object = {one: "one", two: "two", three: "three", name: "조은정"};

let one = object.one;
let two = object.two;
```

```
let three = object.three;

---

let {one, two, three} = object;  // 순서가 아니라 key값을 기준으로 할당됨
let {name: myName, one, two, three} = object;  // name을 myName으로 이름 변경해 할당받을 수 있음


console.log(one, two, three, myName);
```

▼ spread 연산자

```
const cookie = {
  base: "cookie",
  madeIn: "korea"
};

const chocochipCookie = {
  base: "cookie",
  madeIn: "korea",
  toping: "chocochip"
};

const blueberryCookie = {
  base: "cookie",
  madeIn: "korea",
  toping: "blueberry"
};

const strawberryCookie = {
  base: "cookie",
  madeIn: "korea",
  toping: "strawberry"
};
```

```
// 위 코드를 spread 사용해 간편하게 바꾸기
const cookie = {
  base: "cookie",
  madeIn: "korea"
};

const chocochipCookie = {
  ...cookie,
  toping: "chocochip"
};

const blueberryCookie = {
  ...cookie,
  toping: "blueberry"
};

const strawberryCookie = {
  ...cookie,
  toping: "strawberry"
};
```

```
const noTopingCookies = ["촉촉한쿠키", "안촉촉한쿠키"];
const topingCookies = ["바나나쿠키", "블루베리쿠키", "딸기쿠키", "초코칩쿠키"];

const allCookies = [...noTopingCookies, "함정쿠키", ...topingCookies];
```

▼ 동기 & 비동기

- 동기: js에선 이전 작업이 진행 중일 때는 다음 작업 수행 X
  먼저 작성된 코드 실행 끝난 뒤 다음 코드 실행
  - 단점: 하나의 작업이 오래 걸리면 모든 작업이 오래 걸리게 됨
- 비동기: 여러 개의 작업을 동시에 처리

```
// 동기화 방식, taskA 종료 전까지 "코드 끝" 출력되지 않음
function taskA(){
  console.log("A 작업 끝");
}
taskA();
console.log("코드 끝");
```

```
// 비동기 방식
function taskA(){
  setTimeout{()=>{
    console.log("A TASK END");
  }, 2000);
}
taskA();
console.log("코드 끝");  // 코드 끝 먼저 출력 후 A TASK END 출력
```

▼ Promise

- 연속되는 비동기에서 콜백이 끊임없이 이어질 때 이를 해결하기 위한 수단이 Promise
- 비동기 작업이 가질 수 있는 3가지 상태
  - Pending(대기 상태)
  - Fulfilled(성공)
  - Rejected(실패)

```
// 2초 뒤에 전달받은 수가 음수인지 양수인지 판단하는 함수
function isPositive(number, resolve, reject){
  setTimeout(()=>{
    if(typeof number === 'number'){
      // 성공 -> resolve
      resolve(number >= 0 ? "양수":"음수")
    }else{
      // 실패 -> reject
      reject("주어진 값이 숫자형 값이 아닙니다");
```

```
  }, 2000)
}

isPositive(10, (res)=>{
  console.log("성공적으로 수행됨: ", res)
}, (err)=>{
  console.log("실패하였음: ", err);
  }
);
```

```
function isPositive(number, resolve, reject){
  setTimeout(()=>{
    if(typeof number === 'number'){
      // 성공 -> resolve
      resolve(number >= 0 ? "양수":"음수")
    }else{
      // 실패 -> reject
      reject("주어진 값이 숫자형 값이 아닙니다");
  }, 2000)
}

function isPositiveP(number){
  const exector = (resolve, reject) => {
    setTimeout(()=>{
      if(typeof number === "number){
        resolve(number >= 0 ? "양수":"음수");
      }else{
        reject("주어진 값이 숫자형 값이 아닙니다");
      }
    }, 2000);
  };

  const asyncTask = new Promise(executor);  // new 키워드를 사용해 Promise 객체 생성
  return asyncTask;  // isPositiveP의 반환값이 Promise로 바뀜(=해당 함수는 비동기 작업을 한다)
}

const res = isPositiveP(101);
res.then((res)=>{console.log("작업 성공: ", res);}).catch((err)=>{console.log("작업 실패: ", err);})
```

```
// 콜백 지옥 탈출하기
function taskA(a, b, cb){
  return new Promise((resolve, reject) => {
    setTimeout(()=>{
      const res.= a+b;
      cb(res);
    }, 3000)
  });
}

function taskB(a, b){
  return new Promise((resolve, reject) => {
  setTimeout(()=>{
    const res = a*2;
    resolve(res);
  }, 1000);
  });
}

function taskC(a){
  return new Promise((resolve, reject) => {
```

```
  setTimeout(() => {
    const res = a * -1;
    resolve(res);
  }, 2000);
  });
}

taskA(3, 4, (a_res)=>{
  console.log("task A: ", a_res);
  taskB(a_res, (b_res) => {
    console.log("task B: ", b_res);
    taskC(b_res, (c_res) => {
      console.log("task C: ", c_res);
    });
  });
});
```

▼ async & await

- `async function`: 항상 promise 객체 반환
- async로 비동기 처리 가능, 콜백 함수나 promise 객체보다 간편

```
function hello(){
  return 'hello';
}

async function helloAsync(){
  return 'hello Async';
}

helloAsync().then((res)) => {
  console.log(res);
});
```

```
function delay(ms){
  return new Promise((resolve) => {
    setTimeout(resolve, ms);
  });
}

async function getUser(){
  return delay(3000).then(() => {
    return 'hello';
  });
}

getUser.then((res) => {
  const user = res;
  console.log("user: ", user);  // hello 출력
});
```

- await: async 함수 내의 비동기 처리 → 동기로 만듦

```
function delay(ms){
  return new Promise((resolve) => {
```

```
    setTimeout(resolve, ms);
  });
}

async function getUser(){
  await delay(3000);
  return 'hello';
}

getUser().then((res) => {
  const user = res;
  console.log("user: ", user);  // hello 출력
});
```
