

셋째 주

- switch - case
- 배열을 다루는 함수
- 반복문

Switch - Case (switch문)

switch - case 는 if - else 와 비슷한 역할을 합니다.

다만 switch - case의 경우 비교해야할 값을 받고 해당 값에 대한 각각의 상황에 따라 실행되어야 할 로직을 정의합니다.

다음은 switch - case의 문법입니다.

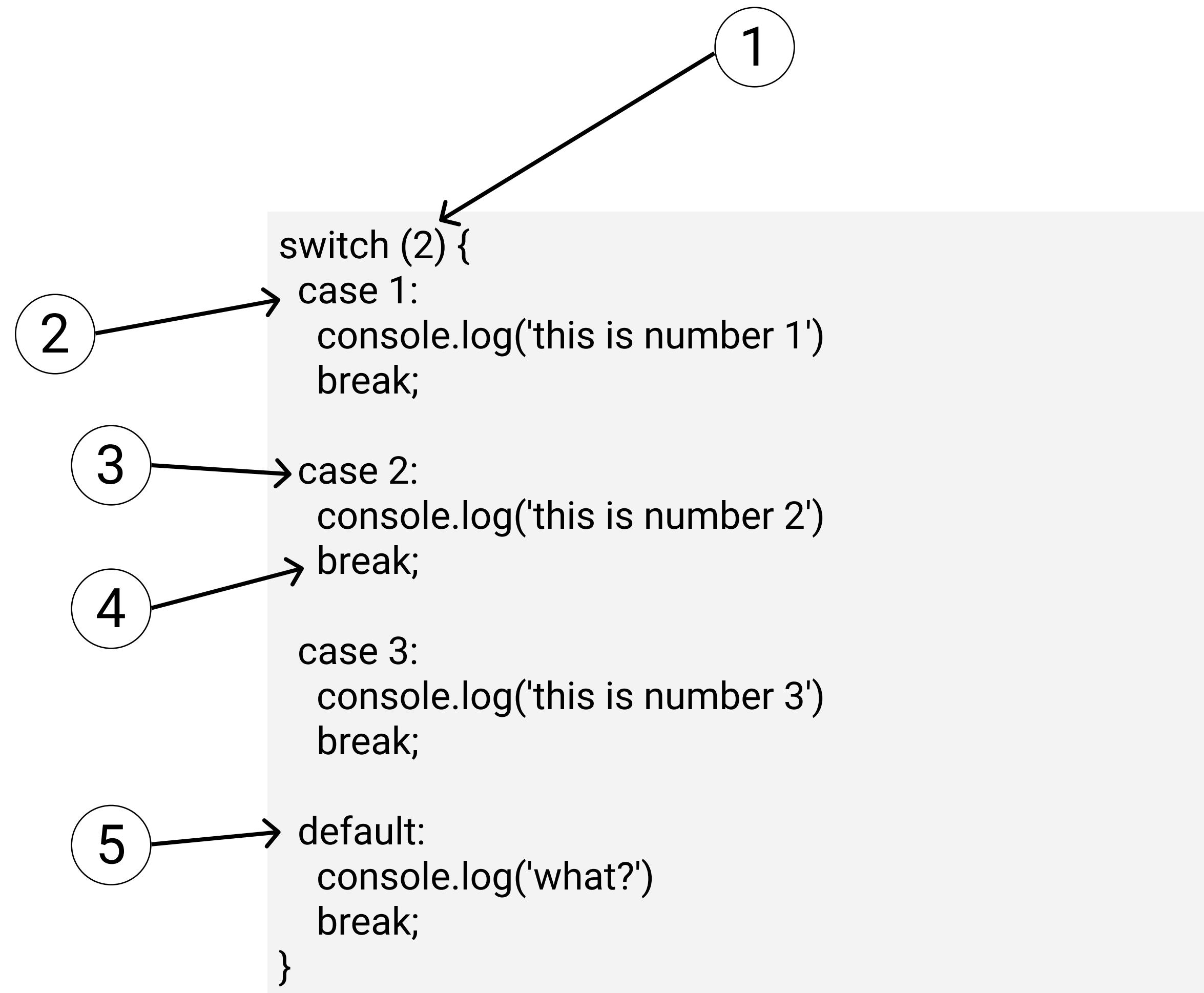
```
switch (비교할 값) {  
    case 비교할 대상:  
        조건 만족시 실행될 로직  
        break;  
  
    default:  
        break;  
}
```

예약어가 많이 사용됩니다. switch, case, default, break 네가지 예약어가 있으며 각각의 의미를 알아봅시다.

Switch - Case (switch문)

- switch
switch 문의 선언부입니다. switch 뒤에는 판별해야할 값을 매개변수로 받습니다.
- case
판별해야할 값을 비교합니다. case 뒤에는 데이터가 정의되어 있으며, 판별해야할 값과 데이터가 같으면 case문 안의 로직이 수행됩니다.
- break
로직을 종료시키는 역할을 합니다.
만약 case 문 안에서 break가 없다면 그 밑의 로직들이 실행됩니다.
- default
만약 판별해야할 값이 case문에 정의된 데이터와 전부 맞지 않으면 실행되는 곳입니다.

Switch - Case (switch문)



예제 코드를 하나씩 풀어보겠습니다.

1. switch 구문을 시작하면서 () 괄호에 값을 넣어주고 있습니다. '2'라는 데이터를 입력받았고 이제 switch 문은 '2'라는 데이터를 비교하기 시작합니다.
2. 첫번째 case를 만났습니다. case 띄고 뒤에 숫자 '1'은 입력받은 '2'와 비교해야 할 대상입니다. 비교를 해보니 '1'과 '2'는 같지 않습니다. 다음으로 넘어가집니다.
3. 두번째 case를 만납니다. 두번째 case가 가진 데이터는 '2'입니다. 비교를 해보니 '2'와 '2'는 같습니다. 이제 이곳의 로직이 실행됩니다.

Switch - Case (switch문)

4. case안의 로직은 console.log 입니다. console.log를 실행하고 다음줄로 넘어갑니다
다음줄에는 break가 있습니다. break 문은 switch-case 문을 끝내는 예약어 이므로 switch-case가 종료됩니다.
5. default 문은 실행되지 않았습니다.
case에 입력된 비교대상들중 입력된 값과 같은 값이 있었기 때문에 실행되지 않았습니다.

문제: default가 실행되려면 어떤 값을 받았어야 했을까요?

switch-case 문은 if-else 문과 상당히 닮았습니다.
그렇기 때문에 사용하는 방법이 둘 중 하나를 선택해야 하는 경우는 잘 없습니다.(거의 취향을 탑니다)
그러므로 각각의 상황에 대해 어떤 코드가 더욱 명시적인지 알 수 있다면 해당 방법을 선택하는게 좋은 방법이라 생각하면 됩니다.

Ex - if-else 를 switch-case로 바꾸기

아래의 numberCheck 함수 안에는 if-else 문이 있습니다.

0| if-else 문을 switch-case 문으로 다시 구현해보세요.

```
function numberChecker(number) {  
  if (number === 1) {  
    console.log('this number is 1')  
  } else if (number === 2) {  
    console.log('this number is 2')  
  } else if (number === 3) {  
    console.log('this number is 3')  
  } else if (number === 4) {  
    console.log('this number is 4')  
  } else {  
    console.log('this is not my number')  
  }  
}
```

배열을 다루는 함수

배열은 여러가지 형태로 가공되는 경우가 많기 때문에

해당 작업들을 도와주는 함수가 존재합니다.

이번에는 매우 간단한 형태의 배열 함수들을 다뤄보겠습니다.(복잡한건 나중에~)

- push
- pop
- shift
- unshift
- splice
- slice
- indexOf
- join

배열을 다루는 함수 - Push, Pop

Push

push 함수는 배열의 마지막에 요소를 추가하는 함수입니다.

push 함수의 매개변수에 데이터를 넣으면 해당 데이터가 배열의 마지막 자리로 들어가게 됩니다.

```
let a = [1,2,3,4,5]  
  
a.push(13)  
console.log(a) // [1,2,3,4,5,13]
```

Pop

pop 함수는 배열의 마지막 요소를 뽑아내는 함수 입니다.

```
let a = [1,2,3,4,5]  
  
let aout = a.pop()  
  
console.log(a) // [1,2,3,4]  
console.log(aout) // 5
```

배열을 다루는 함수 - Shift, Unshift

Shift

shift 함수는 배열의 첫번째 요소를 뽑아내는 함수입니다.

```
let a = [1,2,3,4,5]  
  
let aout = a.shift()  
  
console.log(a) // [2,3,4,5]  
console.log(aout) // 1
```

Unshift

unshift 함수는 배열의 첫번째에 요소를 추가시킵니다.

```
let a = [1,2,3,4,5]  
  
a.unshift(20)  
  
console.log(a) // [20,1,2,3,4,5]
```

배열을 다루는 함수 - Splice, Slice

Splice

splice 함수는 해당 구간의 인덱스 요소를 다른 요소로 바꾸거나 삭제하고 새로운 배열을 반환합니다.

```
let a = [1,2,3,4,5]
let b = a.splice(2,2)

console.log(a) // [1,2,5]
console.log(b) // [3,4]
```

splice의 첫번째 인자값은 제거할 요소의 index를 받습니다.

그리고 두번째 인자값으로 몇개의 요소를 제거할지 받게됩니다.

```
let c = [1, 2, 3, 4, 5]
let d = c.splice(2, 2, 10, 20, 30, 40)

console.log('c: ', c); // [1, 2, 10, 20, 30, 40, 5]
console.log('d: ', d); // [3, 4]
```

배열을 다루는 함수 - Splice, Slice

splice의 세번째 인자값 부터는 제거된 요소의 자리에 새로 들어갈 요소들을 입력하게 됩니다.
콤마(,)를 통해 데이터를 나열하면 해당 데이터들이 순서대로 빈 자리에 들어가게 됩니다.

배열을 다루는 함수 - Splice, Slice

Slice

slice 함수는 해당 위치의 배열을 가져오는 함수입니다.

```
let a = [1,2,3,4,5]
let b = a.slice(1,3)

console.log(a) // [1,2,3,4,5]
console.log(b) // [2,3]
```

splice 와는 다르게 slice는 배열의 요소들을 가져오기 위한 index들의 시작 지점과 종료지점의 index를 인자값으로 받습니다.
그리고 splice의 경우 원본 배열을 변경하지만 slice는 원본 배열이 그대로 유지되는걸 확인할 수 있습니다.

배열을 다루는 함수 - indexOf

IndexOf

indexOf 는 배열의 요소를 탐색하는 함수입니다.

```
let a = [1, 2, 3, 4, 5]
let whereIsFour = a.indexOf(4)

console.log('whereIsFour: ', whereIsFour);
```

indexOf는 배열에서 찾아볼 데이터를 인자값으로 받고 해당 인자값이 있으면 index를 알려줍니다.
만약 해당 값을 찾을 수 없다면 -1 을 반환합니다.

배열을 다루는 함수 - join

join

join은 배열을 string으로 합쳐줍니다.

```
let a = [1, 2, 3, 4, 5]
let joinedA = a.join()
let joinedB = a.join('-')

console.log('joinedA: ', joinedA); // 1,2,3,4,5
console.log('joinedB: ', joinedB); // 1-2-3-4-5
```

그냥 join을 인자값 없이 사용하면 콤마(,) 를 이용해서 모든 원소들을 이어줍니다.

만약 인자값을 넣어주면 해당 인자값을 이용해서 모든 요소들을 이어줍니다.

반복문

javascript에도 다른 언어들과 마찬가지로 반복문이 존재합니다.

하지만 다른 언어들처럼 javascript도 세월이 흐르면서 여러가지 형태의 반복문이 존재합니다.

여기선 가장 기본적인 형태의 반복문만 다뤄보겠습니다.

- while
- for

while

while은 반복문 중에도 조심해서 사용해야 할 반복문입니다.

while은 문법 특성상 무한루프가 걸리기 쉽기 때문입니다.(무한루프: 반복문이 끝나지 않는 상황)

문법은 다음과 같습니다.

```
while(조건) {  
    ....로직  
}
```

문법 자체는 상당히 간단한편입니다.

조건식을 매개변수로 받고 조건이 성립하면 로직이 돌게 됩니다.

이러한 특성 때문에 조건을 잘못 만들게 되면 무한루프에 걸리게 됩니다.

제어문 break, continue

반복문을 제어하는 방법으로 break와 continue가 있습니다.

break는 반복문을 종료시키는 예약어이고

continue는 반복문을 다음 반복으로 넘기는 예약어입니다.

```
let i = 0

while (true) {
  i++
  console.log('ii:', i);

  if (i >= 10) {
    break;
  }
}
```

while의 조건식에 $i \geq 10$ 을 넣어 반복을 제어할 수 있지만

위의 예제처럼 로직 중간에 break를 넣어 반복을 종료시킬 수도 있습니다.

반복 도중에 여러가지 상황에 대해 종료를 시켜야 하는 경우 사용할 수 있습니다.

제어문 break, continue

```
let i = 0

while (i < 10) {
    i++

    if (i % 2 === 0) continue

    console.log('iii:', i)
}
```

continue는 반복문 로직 수행 중간에 로직을 건너뛰는 예약어입니다.

위의 예제는 if문의 조건이 $i \% 2 === 0$ 이므로 if문의 로직이 i 가 짝수일 때 실행됩니다.

if문의 로직이 continue 이므로 if문 밑의 console.log가 수행되지 않고 다음 반복으로 넘어갑니다.

결과적으로 해당 로직은 홀수만 콘솔에 찍히게 됩니다.

Ex - 반복문 while

1~100 까지 숫자에서 짝수만 더하는 반복문을 만들어주세요

- while문 사용
- while문 조건식은 true
- 로직 중간에 i가 100이 넘어가면 break 수행
- i는 1씩 증가
- i가 짝수일 경우 evenSum 변수에 합해줌
- evenSum에 짝수의 합이 저장됨

for

for문은 조금 복잡해보이는 반복문입니다.

하지만 while문에서 필요한 부분들을 단축시킨 형태의 문법으로 이해하시면 조금 더 쉽게 이해하실수 있습니다.

```
for(시작; 조건; 로직후 실행됨) {  
    ....로직  
}
```

- 시작

기존에 우리가 while을 사용하기위한 index 변수 i를 선언했었습니다.

for문에서는 이런 index변수를 선언하기 위한 자리를 제공하는데 시작 자리에 변수 선언을 해주면 됩니다.

- 조건

for문의 로직이 수행되는 조건을 입력합니다.

true일 경우 로직이 계속 수행되는데, 보통 i를 통해서 조건을 제어합니다.

- 로직후 실행됨

로직 후 실행되어야 할 구문이 들어가는 부분입니다.

보통 index를 조절하는 구문을 넣어줍니다.

for

```
for (let i = 0; i < 10; i++) {  
  console.log('i: ', i);  
}
```

로직 자체는 많이 짧습니다. 하나 하나씩 해당 로직에 대해 알아보겠습니다.

- 시작 부분에 `let i = 0` 를 선언해줬습니다.
`i`라는 변수를 `index` 변수로 사용하기 위해 선언해줬습니다. 이 변수 `i`는 `for`문 안에서만 사용 가능합니다.
- 조건에 `i < 10`이라는 조건을 걸어줬습니다. 지금은 `i == 0` 이므로 조건이 참이되며 `for`문 안의 로직이 동작하게 됩니다.
- `for`문 안의 로직이 돌고 나면 `i++`가 동작하게 됩니다.

이제 `i == 1`되고 `for`문은 다시 조건을 확인합니다.

이번에도 `i < 10` 이므로 로직이 또 동작하게 됩니다.

이러한 반복은 `i < 10` 조건이 성립하지 않을 때 까지 반복하게 됩니다.

Ex - 반복문 for

1~100 까지 숫자중 홀수만 들어있는 배열을 만들어 보세요

- for문 사용
- 변수명 oddList로 변수를 선언해서 1~100 사이의 홀수만 들어있는 배열을 만들어 보세요
- push를 사용해주세요