

16. 자바스크립트 기본 문법



16-1 변수 알아보기

16-2 자료형 이해하기

16-3 연산자 알아보기

16-4 조건문 알아보기

16-5 반복문 알아보기

변수 알아보기

변수란

변수(variable) : 프로그램에서 자료를 담아두는 공간

- **재사용성**: 한 번 저장한 값 재사용 가능

(예) 학생의 점수를 변수에 저장하면, 그 점수를 계산하거나 출력할 때마다 다시 입력할 필요가 없음.

- **가독성**: 코드를 읽기 쉽다.

(예) `let studentScore = 90;` 이라고 쓰면, 이 코드를 보는 사람은 `studentScore`가 학생의 점수를 의미한다는 것을 쉽게 알 수 있음

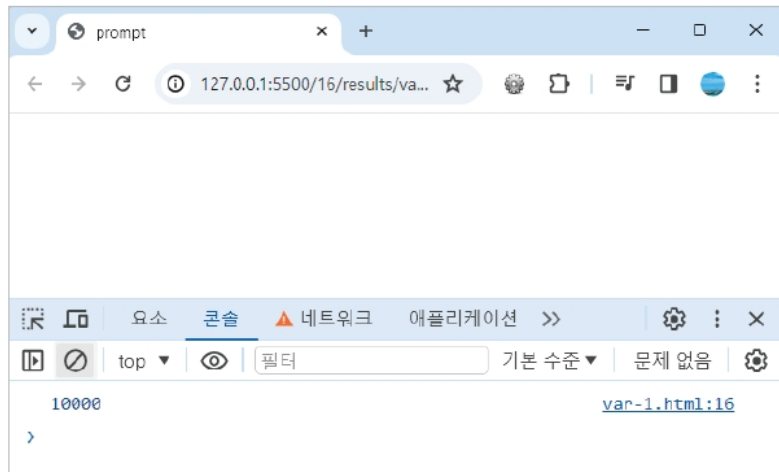
변수 선언하기

let 뒤에 변수 이름 작성

기본형 let 변수명

(예) 사각형의 넓이 구하기

```
let width; /* 너비를 저장할 변수 선언 */
let height; /* 높이를 저장할 변수 선언 */
width = 200; /* 변수에 값 대입 */
height = 50; /* 변수에 값 대입 */
let area = width * height; /* 변수 선언과 동시에 계산 결과값 대입 */
console.log(area); /* 사각형의 넓이를 콘솔에 출력 */
```



변수 알아보기

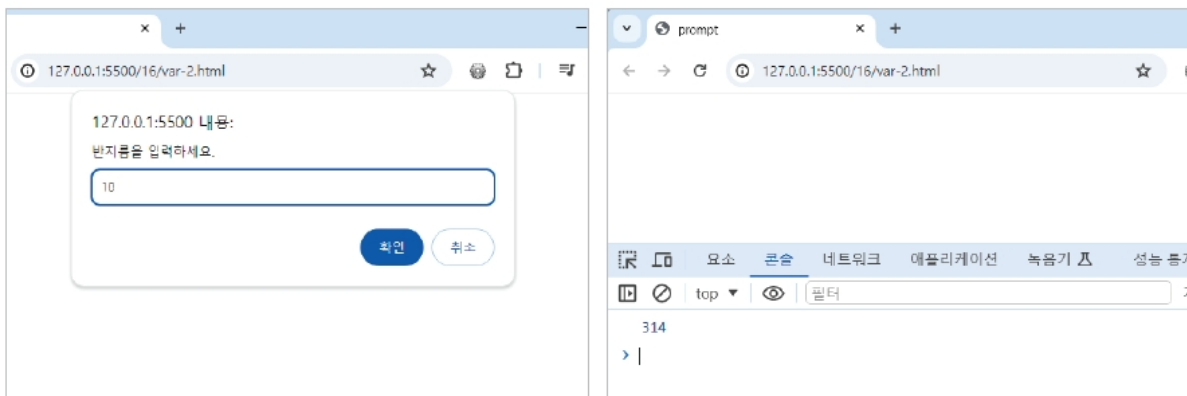
상수 선언하기

- `const` 뒤에 변수 이름 작성
- 상수는 변수의 일종. 한번 값을 할당하면 프로그램 안에서 값이 바뀌지 않음

기본형 `const` 변수명

(예) 원의 넓이 구하기

```
const PI = 3.14; /* 원주율을 상수로 선언 */
let radius = prompt("반지름을 입력하세요."); /* 반지름 입력 받기 */
let area = PI * radius * radius; /* 변수를 사용해 원의 넓이 계산 */
console.log(area); /* 원의 넓이를 알림 창에 출력 */
```



변수 선언의 규칙

- 변수 이름
 - 영어 문자, 언더스코어(_), 숫자를 사용한다
 - 첫 글자는 영문자, _기호, \$기호를 사용한다
 - 띄어쓰기나 기호는 허용하지 않는다
 - 예) now, _now, now25 (사용할 수 있음)
 - 예) 25now, now 25, *now (사용할 수 없음)
- 영어 대소문자를 구별하며 예약어는 변수 이름으로 사용할 수 없다
- 여러 단어를 연결할 때는 하이픈이나 언더스코어를 사용할 수 있고 중간에 대문자를 섞어 쓸 수도 있다
 - 예) total-area, total_area, totalArea 등
- 변수 이름은 의미있게 작성한다

자료형 이해하기

자료형이란

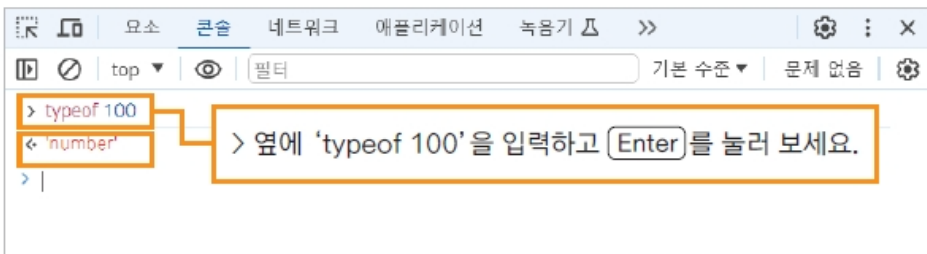
컴퓨터가 처리할 수 있는 자료의 형태

종류		설명	예시
원시 유형	숫자형	따옴표 없이 숫자로만 표기합니다.	<code>let birthYear = 2000;</code>
	BigInt	숫자형으로는 표현할 수 없는 아주 큰 정수로. 숫자 끝에 n을 붙여서 표시합니다.	<code>let bigIntNumber = 12345678901234567890...n</code>
	문자열	작은따옴표(' ')나 큰따옴표(" ")로 묶어서 나타냅니다. 숫자를 따옴표로 묶으면 문자로 인식합니다.	<code>let greeting = "Hello!"; let birthYear = "2000";</code>
	논리형	참(true)과 거짓(false)이라는 2가지 값만 있는 유형입니다. 이때 true와 false는 소문자로만 표시합니다.	<code>let isEmpty = true;</code>
복합 유형	배열	하나의 변수에 값을 여러 개 저장합니다.	<code>let seasons = ['봄', '여름', '가을', '겨울'];</code>
	객체	함수와 속성을 함께 포함합니다.	<code>let date = new Date();</code>
특수 유형	undefined	자료형이 지정되지 않았을 때의 상태입니다. 예를 들어 변수 선언만 하고 값을 할당하지 않은 변수는 undefined 상태입니다.	
	null	값이 유효하지 않을 때의 상태입니다.	

자료형 이해하기

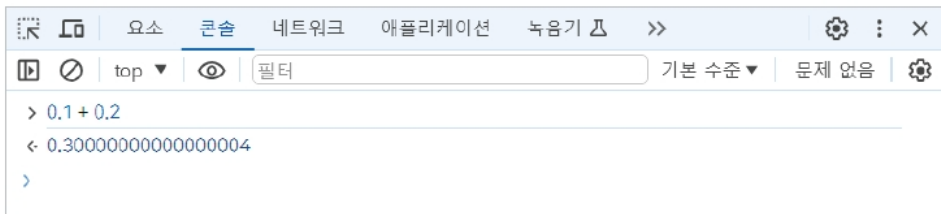
숫자형(number)

- 정수 : 소수점 없는 숫자



콘솔 창에서 typeof 다음에 숫자를 입력하면 number라고 표시됨

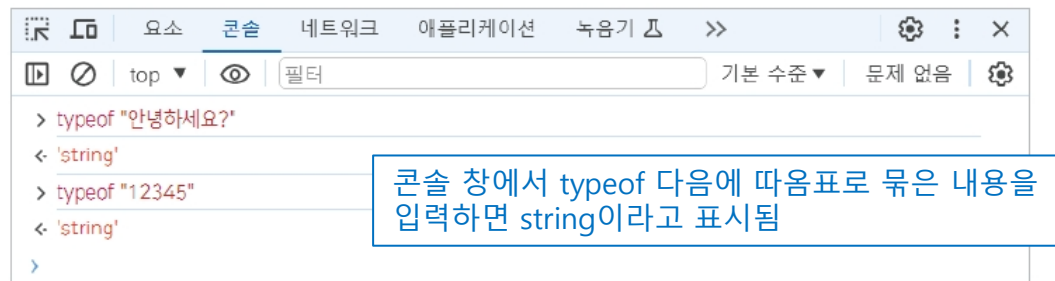
- 실수 : 소수점이 있는 숫자



※ 자바스크립트는 실수를 정밀하게 계산하지 못함

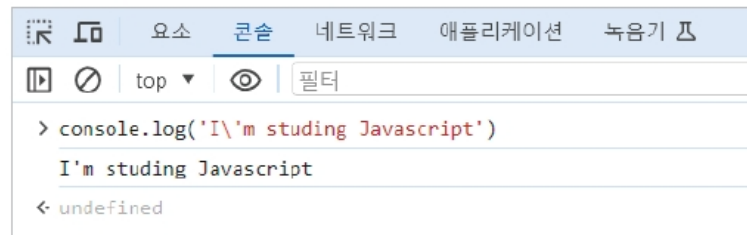
문자열(string)

- 작은따옴표(' ')나 큰따옴표(" ")로 묶은 데이터



- 특수 기호를 표시할 때는 백슬래시(\) 다음에 기호 입력

```
console.log('I\'m studing Javascript')
```

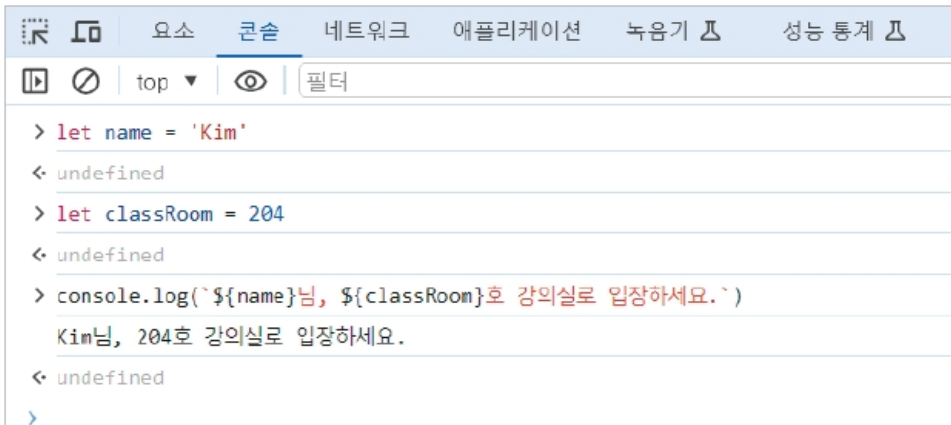


자료형 이해하기

템플릿 리터럴(template literal)

- 문자열과 변수, 식을 섞어서 하나의 문자열을 만드는 방식
- 백틱(`) 기호를 사용해 문자열을 만든다.
- 템플릿 리터럴 안의 변수나 식은 \${ } 로 묶어서 표현
- 태그나 띄어쓰기, 특수 기호(이스케이프 문자)는 그대로 사용 가능

```
let name = 'Kim'
let classRoom = 204
console.log(`${name}님, ${classRoom}호 강의실로 입장하세요.`)
```



```
> let name = 'Kim'
< undefined
> let classRoom = 204
< undefined
> console.log(`${name}님, ${classRoom}호 강의실로 입장하세요.`)
Kim님, 204호 강의실로 입장하세요.
< undefined
>
```

논리형(boolean)

- 참true이나 거짓false의 값을 표현하는 자료형. 불린 유형이라고도 함.
- 조건을 확인해서 조건이 맞으면 true, 맞지 않으면 false라는 결괏값 출력

undefined 유형

- 자료형이 정의되지 않았을 때의 데이터 상태
- 변수 선언만 하고 값이 할당되지 않은 자료형

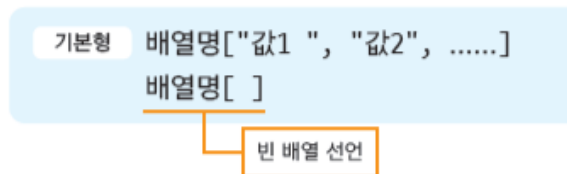
null 유형

- 데이터 값이 유효하지 않은 상태
- 변수에 할당된 값이 유효하지 않다는 의미

자료형 이해하기

배열(array)

하나의 변수에 여러 값을 저장할 수 있는 복합 유형

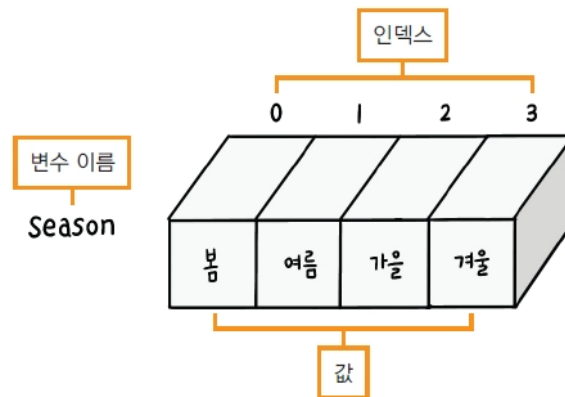
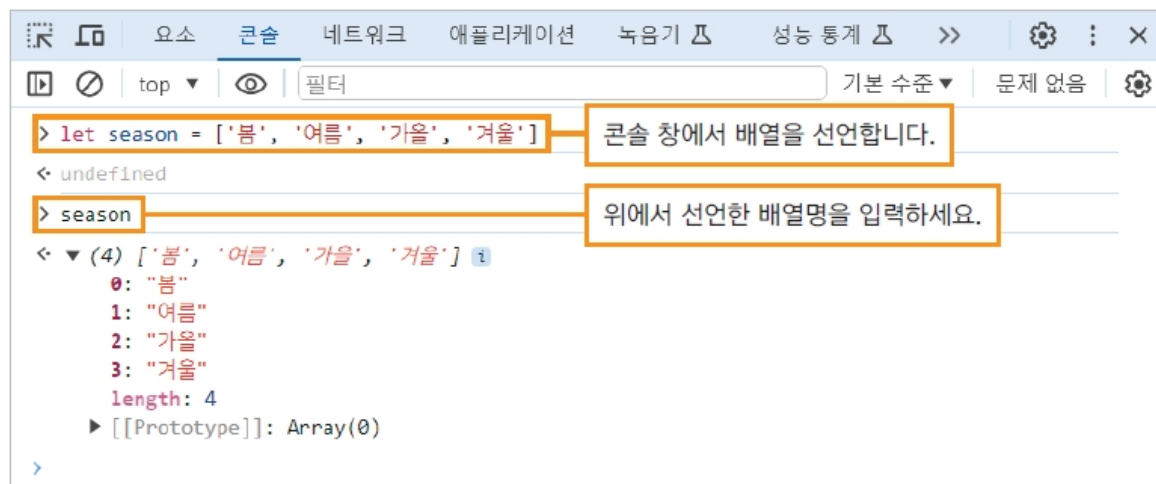


예) 배열을 사용하지 않는다면

```
let spring = "봄";  
let summer = "여름";  
let fall = "가을";  
let winter = "겨울";
```

예) 배열을 사용한다면

```
let season = ["봄", "여름", "가을", "겨울"];
```



자료형 변환하기

느슨한 자료형 체크

- 자바스크립트는 미리 변수의 자료형을 지정하지 않는다.
- 변수에 값을 할당하면 그 시점에 자료형이 결정된다.
- 여러 사람이 프로젝트를 진행할 경우 변수를 일관성 있게 유지하기 힘들다.

```
num = 20 // 숫자형  
num = 'John' // 문자열
```

자동 형 변환

- 연산하는 동안 자동으로 자료형이 바뀔 수 있다.
- 문자열 값을 사칙 연산에서 사용하면 자동으로 숫자형으로 변환됨.
- 숫자를 문자열에 연결하면 자동으로 문자열로 변환됨

```
input = prompt("숫자를 입력하세요.") // input은 문자열  
input * 10 // 50. input이 숫자형으로 바뀜
```

```
let one = '20' // 문자열  
let two = 10 // 숫자형  
one + two // '2010'. 숫자였던 two가 문자열로 바뀜
```


자료형 변환하기

자료형 변환 함수

함수	설명
Number()	문자열이나 논리형 값을 숫자로 변환합니다.
parseInt()	문자열을 정수 숫자로 변환합니다.
parseFloat()	문자열을 실수 숫자로 변환합니다.
String()	숫자나 논리형 값을 문자열로 변환합니다.
Boolean()	괄호 안의 값을 논리형으로 변환합니다.

```
Number('123') // 123
```

```
Number('123ABC') // NaN
```

```
parseInt('123') // 123
```

```
parseInt('123.45') // 123
```

```
parseInt('123ABC') // 123
```

```
parseFloat('123') // 123
```

```
parseFloat('123.45') // 123.45
```

```
parseFloat('123ABC') // 123
```

```
String(123) // '123'
```

```
String(true) // 'true'
```

```
Boolean(1) // true
```

```
Boolean(0) // false
```

```
Boolean('ABC') // true
```

```
Boolean('') // false
```

연산자 알아보기

산술 연산자

수학 계산을 할 때 사용하는 연산자

종류	설명	예시
+	두 피연산자의 값을 더합니다.	$c = a + b$
-	첫 번째 피연산자 값에서 두 번째 피연산자 값을 뺍니다.	$c = a - b$
*	두 피연산자의 값을 곱합니다.	$c = a * b$
/	첫 번째 피연산자 값을 두 번째 피연산자 값으로 나눕니다.	$c = a / b$
%	첫 번째 피연산자 값을 두 번째 피연산자 값으로 나눈 나머지를 구합니다.	$c = a \% b$
++	피연산자를 1 증가시킵니다.	$a++$
--	피연산자를 1 감소시킵니다.	$b--$

나누기 연산자(/) : 나눈 값 자체

나머지 연산자(%) : 나눈 후에 남은 나머지 값

```
let numberOne = 15 / 2; // 7
let numberTwo = 15 % 2; // 1
```

할당 연산자(대입 연산자)

연산자 오른쪽의 실행 결과를 왼쪽 변수에 할당하는 연산자

종류	설명	예시
=	연산자 오른쪽의 값을 왼쪽 변수에 할당합니다.	$y = x + 3$
+=	$y = y + x$ 를 의미합니다.	$y += x$
-=	$y = y - x$ 를 의미합니다.	$y -= x$
*=	$y = y * x$ 를 의미합니다.	$y *= x$
/=	$y = y / x$ 를 의미합니다.	$y /= x$
%=	$y = y \% x$ 를 의미합니다.	$y \% = x$

연산자 알아보기

비교 연산자

피연산자 2개의 값을 비교해서 true나 false로 결과값 반환

종류	설명	예시	
		조건식	결과값
==	피연산자가 서로 같으면 true입니다.	3 == "3"	true
===	피연산자도 같고 자료형도 같으면 true입니다.	a === "3"	false
!=	피연산자가 서로 같지 않으면 true입니다.	3 != "3"	false
!==	피연산자가 같지 않거나 자료형이 같지 않으면 true입니다.	3 !== "3"	true
<	왼쪽 피연산자가 오른쪽 피연산자보다 작으면 true입니다.	3 < 4	true
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같으면 true입니다.	3 <= 4	true
>	왼쪽 피연산자가 오른쪽 피연산자보다 크면 true입니다.	3 > 4	false
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같으면 true입니다.	3 >= 4	false

== 연산자 와 != 연산자

피연산자의 자료형을 자동으로 변환해서 비교

```
3 == "3" // true
3 != "3" // false
```

=== 연산자 와 !== 연산자

← 프로그램에서 값을 비교할 때 더 많이 사용

피연산자의 자료형을 변환하지 않음

```
3 === "3" // false
3 !== "3" // true
```

문자열의 비교

- 문자열에 있는 문자들의 ASCII 값을 비교
- 비교할 문자가 여러 개일 경우 맨 앞의 문자부터 하나씩 비교

연산자 알아보기

논리 연산자

true와 false가 피연산자인 연산자
조건을 처리할 때 사용

종류	기호	설명
OR 연산자		피연산자 중 하나만 true여도 true가 됩니다.
AND 연산자	&&	피연산자가 모두 true일 경우에만 true가 됩니다.
NOT 연산자	!	피연산자의 반댓값을 지정합니다.

조건문 알아보기

if 문과 if ~ else 문

피연산자 2개의 값을 비교해서 true나 false로 결괏값 반환
하나의 if ~ else 문 안에 다른 if ~ else 문을 넣을 수 있다

```
기본형 if (조건) {  
    조건 결괏값이 true일 때 실행할 명령  
}
```

```
기본형 if (조건) {  
    조건 결괏값이 true일 때 실행할 명령  
} else {  
    조건 결괏값이 false일 때 실행할 명령  
}
```

(예) 3의 배수 확인하기 1

```
<script>  
    let userNumber = parseInt(prompt('숫자를 입력하세요.'));  
    if (userNumber % 3 === 0)  
        alert('3의 배수입니다.');    else  
        alert('3의 배수가 아닙니다.');</script>
```

127.0.0.1:5500 내용:

숫자를 입력하세요.

14

확인

취소

127.0.0.1:5500 내용:

3의 배수가 아닙니다.

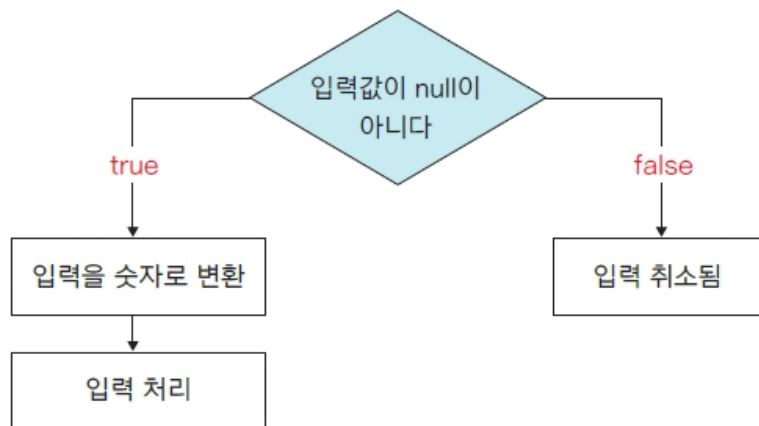
확인

조건문 알아보기

중첩된 if ~ else 문

3의 배수 확인하기 프로그램에서 프롬프트 창에서 [취소] 눌렀을 때도 고려하기

- [취소] 버튼을 눌렀는지 확인하고 (값이 null인지 체크)
- (아닐 경우) [확인] 버튼 눌렀을 때 3의 배수 체크하기



(예) 3의 배수 확인하기 2

```
<script>
let userNumber = prompt('숫자를 입력하세요.');
```



```
if (userNumber !== null) {
  if (parseInt(userNumber) % 3 === 0)
    alert('3의 배수입니다.');
```



```
  else
    alert('3의 배수가 아닙니다.');
```



```
}
else
  alert('입력이 취소되었습니다.');
```



```
</script>
```

조건문 알아보기

조건 연산자로 조건 체크하기 ← '삼항 연산자' 라고도 함

조건이 하나이고 true, false일 때 실행할 명령이 각각 1개 뿐일 때 간단하게 사용할 수 있음

기본형 (조건) ? true일 때 실행할 명령 : false일 때 실행할 명령

(예) 3의 배수 확인하기 3

```
<script>
  let userNumber = prompt('숫자를 입력하세요.');
```



```
  if (userNumber !== null) {
    (parseInt(userNumber) % 3 === 0) ? alert('3의 배수입니다.') : alert('3의 배수가 아닙니다.');
```



```
  }
  else
    alert('입력이 취소되었습니다.');
```



```
</script>
```

```
if (parseInt(userNumber) % 3 === 0)
  alert('3의 배수입니다.');
```



```
else
  alert('3의 배수가 아닙니다.');
```

조건문 알아보기

논리 연산자로 조건 체크하기

- 조건을 2개 이상 체크할 경우에는 조건 연산자를 사용해 조건을 만들
- 두 조건이 true일 경우, 조건 1개만 true일 경우처럼 여러 경우를 따질 때 논리 연산자 사용

AND 연산자 (&&)

피연산자 2개 중에서 false가 하나라도 있으면
결괏값은 false

op 1	op 2	op 1 && op 2
false	false	false
false	true	false
true	false	false
true	true	true

OR 연산자 (||)

피연산자 2개 중에서 true가 하나라도 있으면
결괏값은 true

op 1	op 2	op 1 op 2
false	false	false
false	true	true
true	false	true
true	true	true

NOT 연산자 (!)

피연산자를 반대로 뒤집음

op	!op
false	true
true	false

조건문 알아보기

switch문

- 처리할 명령이 많을 경우 switch 문이 편리

```
기본형 switch (조건)
{
  case 값1: 명령1
    break
  case 값2: 명령2
    break
  .....
  default: 명령n
}
```

- 조건은 case 문의 값과 일대일로 일치해야 함
- case 문의 명령 실행 후 switch 문 빠져나옴
- 조건과 일치하는 case 문이 없다면 default 문 실행
- default 문에는 break 문을 사용하지 않아도 됨

(예) switch문으로 조건 체크하기

```
<script>
let session = prompt("관심 세션을 선택해 주세요. 1-마케팅, 2-개발, 3-디자인");
switch(session) {
  case "1": document.write("<p>마케팅 세션은 <strong>201호</strong>에서 ..... </p>")
    break;
  case "2": document.write("<p>개발 세션은 <strong>203호</strong>에서 ..... </p>")
    break;
  case "3": document.write("<p>디자인 세션은 <strong>205호</strong>에서 ..... </p>")
    break;
  default: alert("잘못 입력했습니다."); // 1, 2, 3이 아닌 값을 입력받으면 출력
}
</script>
```

127.0.0.1:5500 내용:

관심 세션을 선택해 주세요. 1-마케팅, 2-개발, 3-디자인

확인 취소

개발 세션은 **203호**에서 진행됩니다.

반복문 알아보기

for 문



- 1 초깃값: 카운터 변수 초기화. 초깃값은 0이나 1부터 시작하는게 대부분.
- 2 조건: 명령을 반복하기 위해 조건 체크.
이 조건을 만족해야 그다음에 오는 명령을 실행할 수 있다.
- 3 증가식: 명령을 반복한 후 실행. 보통 카운터 변수를 1 증가시키는 용도로 사용.

(예) for문을 사용해 1부터 5까지 숫자 더하기

```
<script>  
let sum = 0;  
for(let i = 1; i < 6; i++) {  
    sum += i;  
}  
document.write(`1부터 5까지 더하면 ${sum}`);  
</script>
```

for 문은 초깃값 → 조건 체크 → 명령 실행 → 증가식 → 조건 체크 → 의 순서로 진행됨

- 1 카운터로 사용할 변수 i에 초깃값 1 지정
- 2 i = 1 → i < 6 체크 → (조건 만족함) → sum += i 실행 → i++ 실행
- 3 i = 2 → i < 6 체크 → (조건 만족함) → sum += i 실행 → i++ 실행
- 4 i = 3 → i < 6 체크 → (조건 만족함) → sum += i 실행 → i++ 실행
- 5 i = 4 → i < 6 체크 → (조건 만족함) → sum += i 실행 → i++ 실행
- 6 i = 5 → i < 6 체크 → (조건 만족함) → sum += i 실행 → i++ 실행
- 7 i = 6 → i < 6 체크 → (조건 만족하지 않음) → for 문을 빠져나옴

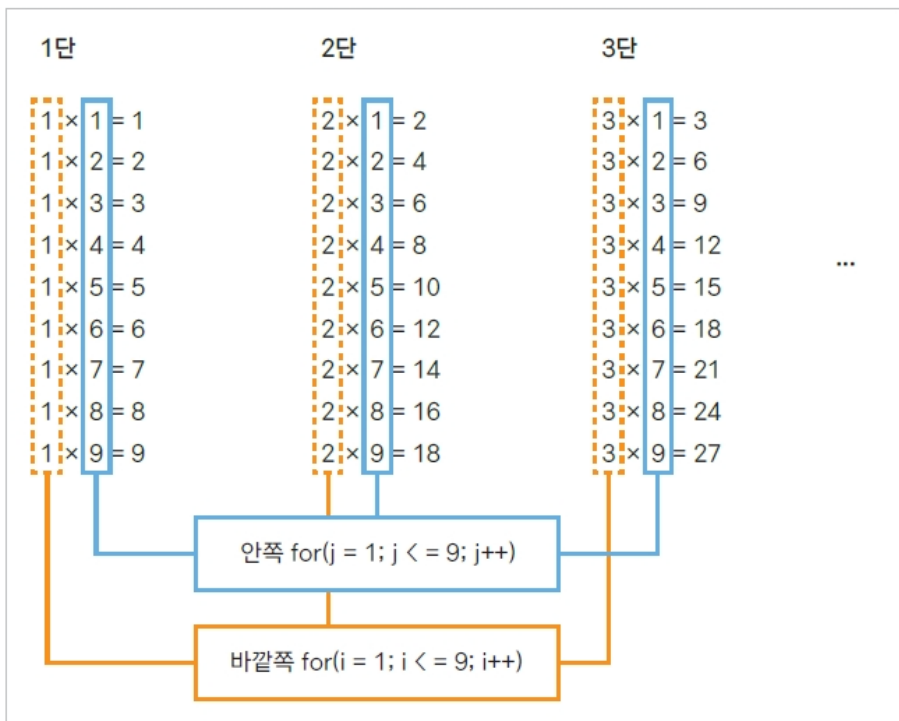
반복문 알아보기

중첩된 for 문

for문 안에 또다른 for문 사용하는 방법

안쪽 for문을 모두 실행한 후 바깥쪽 for문 실행

(예) 중첩된 for문을 사용해 구구단 프로그램 만들기



```
<script>
```

```
for (let i = 1; i <= 9; i++) {  
  document.write(`<h3>${i}단</h3>`);  
  for (let j = 1; j <= 9; j++) {  
    document.write(`${i} X ${j} = ${i * j} <br>`);  
  }  
}
```

```
</script>
```

구구단

1단

1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
1 X 4 = 4
1 X 5 = 5
1 X 6 = 6
1 X 7 = 7
1 X 8 = 8
1 X 9 = 9

2단

2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18

3단

3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
...

while 문과 do ~ while 문

while 문

조건을 체크하고 true라면 { }안의 명령 실행

→ 조건이 false라면 명령은 한 번도 실행하지 않을 수 있음

```
기본형 while (조건) {  
    실행할 명령  
}
```

do ~ while 문

일단 명령을 한번 실행한 후 조건 체크.

true라면 { } 안의 명령 실행, false라면 { }을 빠져나옴

→ 조건이 false라도 명령은 최소한 한 번은 실행

```
기본형 do {  
    실행할 명령  
} while (조건)
```

(예) while문으로 * 표시하기

```
<script>  
  let stars = parseInt(prompt('몇 개의 별을 표시할까요?'));  
  while (stars > 0) {  
    document.write('*');  
    stars--;  
  }  
</script>
```

(예) do...while문으로 * 표시하기

```
<script>  
  let stars = parseInt(prompt('몇 개의 별을 표시할까요?'));  
  do {  
    document.write('*');  
    stars--;  
  } while (stars > 0);  
</script>
```

stars 값을 0으로 지정하면?

- while 문 : 아무것도 표시되지 않음
- do...while 문 : 별 1개 표시됨

break 문과 continue 문

break 문

종료 조건이 되기 전에 반복문을 빠져 나와야 할 때 사용

기본형 break

(예) 구구단 2단~4단만 표시하기

```
<div class="container">
<script>
  for (let i = 2; i <= 9; i++) {
    document.write(`<div class="times">`);
    document.write(`<h3>${i}단</h3>`);
    for (let j = 1; j <= 9; j++) {
      document.write(`${i} X ${j} = ${i * j} <br>`);
    }
    document.write(`</div>`);
    if (i === 4) { // 4단까지만 표시
      break;
    }
  }
</script>
</div>
```

continue 문

조건에 해당되는 값을 만나면 반복문의 맨 앞으로 이동

→ 결과적으로 반복 과정을 한 차례 건너 뛴

기본형 continue

(예) 1부터 10까지 짝수만 더하기

```
<script>
let n = 10;
let sum = 0;
for (let i = 1; i <= n; i++) {
  if (i % 2 === 1) { // 홀수는 더하지 않고 건너뛴
    continue
  }
  sum += i;
  document.write(`${i} ----- ${sum} <br>`);
}
</script>
```