



02



## 02-4. Control Flow



JavaScript Basic

# if

---

## 제어문

- 제어문이란 프로그램의 실행 흐름을 제어하기 위한 구문입니다.
- 제어문은 어떤 특정 영역의 구문을 조건에 만족하는 경우에만 실행시키고자 할 때 혹은 반복적으로 실행시키고자 할 때 사용하는 기법입니다.
- 조건문은 참 거짓이 나오는 조건을 명시하고 어떤 조건이 참인 경우에만 특정 코드가 실행되게 하고자 할 때 사용되는 구문입니다.
- 조건문은 크게 if 문과 switch문 그리고 3항 연산자가 있습니다.

# if

---

## if문

- 조건문의 가장 대표적이고 가장 많이 사용되는 것이 if 문입니다.
- if 문은 조건이 true 인 경우에만 실행되는 코드를 묶기 위해서 사용됩니다.

```
if( 조건 ) {  
    조건에 만족하는 경우 실행되어야 하는 구문  
}
```

# if

---

## if문

- 조건을 명시하는 ( ) 부분은 true 혹은 false 값이 적용이 되어야 합니다.
- 논리 타입의 값은 true, false 이지만 자바스크립트에서는 숫자, 문자, null, undefined 도 논리타입인 true, false 로 이용될 수 있습니다.
- 그럼으로 if( ) 에 0, null, undefined 가 조건으로 지정되면 false 로 판단되며 1, "hello" 등의 데이터가 지정되면 true 로 판단합니다.

```
if 조건 명시
1  if(1 && 'hello'){
2    console.log('if 문이 실행됩니다..')
3  }
4  if(0 || null || undefined){
5    console.log('if 문이 실행되지 않습니다..')
6  }
```

# if

---

## if문

- 만약 조건에 만족했을 때 실행되는 영역이 1줄이라면 {} 을 생략할 수도 있습니다.

{ } 생략
1 <code>if(age &lt;= 30)</code>
2 <code>console.log('실행 영역이 1 줄인 경우 {} 생략가능')</code>

# if

---

## if – else

- 조건에 만족하지 않는 경우에 실행시켜야 하는 코드도 있습니다.
- 이 경우에는 else 예약어를 이용해 조건에 만족하지 않은 경우에 실행할 코드를 명시합니다.

```
if( 조건 ){  
    조건에 만족하는 경우 실행되어야 하는 구문  
}else {  
    조건에 만족하지 않는 경우 실행되어야 하는 구문  
}
```

- else 는 독립적으로 사용될 수는 없습니다. 항상 if에 조건이 명시되어야하고 그 조건에 만족하지 않는 경우에 실행될 코드를 else 에 담습니다.
- else 부분도 else { } 형태처럼 실행시켜야 하는 부분을 { } 로 묶어야 하는데 만약 실행시켜야 하는 코드가 1 줄이라면 { } 을 생략할 수도 있습니다.

# if

---

## if – else if – else

- 어떤 로직이 실행되어야 하는 경우 조건을 여러 번 주어야 하는데 이때는 else if( ) 를 사용합니다.
- else if 를 사용하려면 먼저 if 가 먼저 선언되어 있어야 하고 그 조건에 만족하지 않는 경우 다른 조건을 명시하고자 할 때 사용합니다.

```
if( 조건 ) {  
    }else if( 조건 ) {  
    }else if( 조건 ) {  
    }else {  
    }  
}
```

- else if 뒤에 마지막으로 else 가 추가될 수도 있고 추가되지 않을 수도 있습니다.



## if 테스트

---

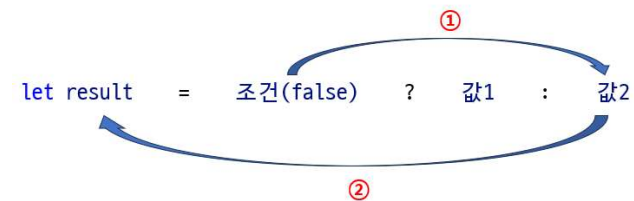
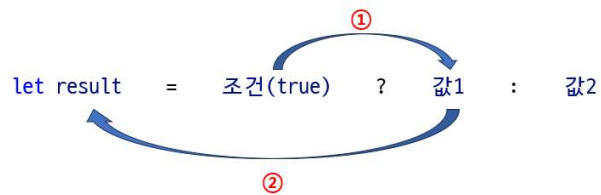
**다음의 조건대로 코드를 작성하세요!!**

- 유저에게 `prompt()` 를 이용해 나이를 입력받는다.
- `prompt` 에서 취소 버튼을 누르면 (`null` 이면) '입력을 취소 하였습니다.' 를 `console` 로 출력한다.
- 숫자가 아닌 데이터를 입력하면 ( `isNaN(data)` 가 `true` 이면 ) '숫자가 아닌 문자를 입력 하셨습니다.' 를 `console` 로 출력한다.
- 0 미만, 100 이상의 숫자를 입력하면 '1 이상, 100 이하의 숫자를 입력해야 합니다.' 를 `console` 로 출력한다.
- 10 미만의 숫자를 입력하면 '어린이군요' 를 `console` 로 출력한다.
- 10 이상, 20 미만의 숫자를 입력하면 '청소년이군요' 를 `console` 로 출력한다.
- 20 이상의 숫자를 입력하면 '어른군요' 를 `console` 로 출력한다.
- 이 코드를 `if` 문을 이용해 작성하세요.



## 3항 연산자

- 3항 연산자는 연산자입니다.
- 연산자라 함은 무언가 연산이 실행되고 그 연산에 의한 결과가 나오게 됩니다.
- 3항 연산자도 결과가 나오지만 조건을 명시할 수 있고 그 조건에 따라 다르게 결과가 나오게 하는 연산자입니다.



## switch - case

---

- 조건문을 작성할 때 switch - case 구문을 사용할 수도 있습니다.
- switch - case 문은 어떤 데이터의 값이 여러 개가 나올 수 있는데 그 값이 어떤 것인지에 따라 실행되는 구문을 다르게 조건을 주고자 할 때 사용됩니다.

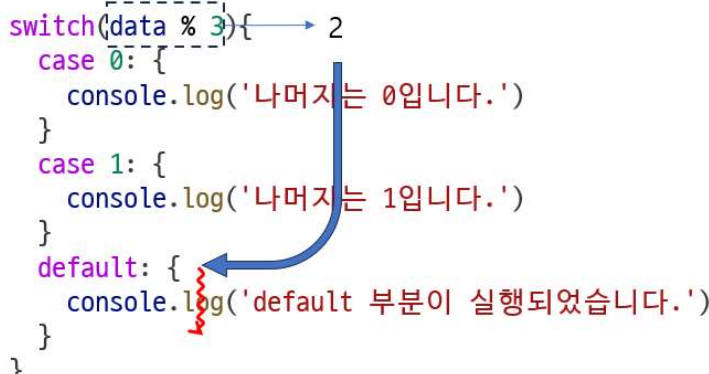
```
switch(데이터){  
  case 값1: {  
    실행 구문  
  }  
  case 값2: {  
    실행 구문  
  }  
  default: {  
    실행 구문  
  }  
}
```

- switch - case 문을 사용한다면 switch( ) 에는 어떤 데이터가 명시되어야 합니다.
- 데이터의 값에 대한 조건은 case 예약어로 명시합니다.
- case 뒤에 값을 명시하고 : 뒤에 실행로직을 { } 로 명시합니다.
- case 를 여러 개 나열하여 switch 에 명시된 데이터 값에 따라 맞는 case 부분이 실행되게 작성합니다.

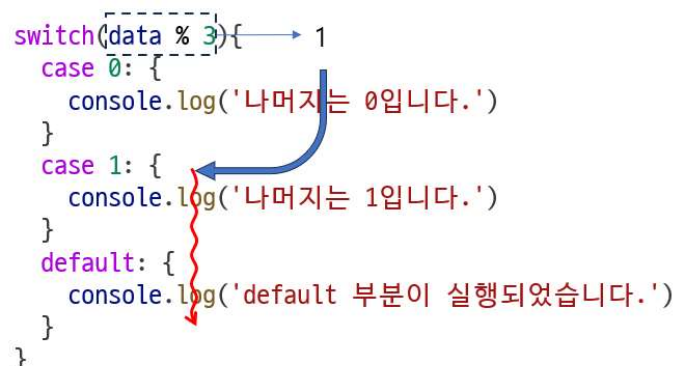
## switch - case

- case 의 맨 마지막에 default 가 작성될 수 있습니다.
- default 는 생략이 가능하며 작성한다면 case 가 나열되고 맨 마지막에 작성됩니다.
- default 는 위에 선언된 case 값에 만족하지 않는 경우 실행될 구문을 명시하기 위해서 사용됩니다.
- switch - case 문에서 만약 case 가 여러 개 있는 경우 윗부분 case 값 조건에 만족하게 되면 그 위치부터 아래에 선언된 모든 case 와 default 부분이 실행되게 됩니다.

```
switch(data % 3){  
  case 0: {  
    console.log('나머지는 0입니다.')  
  }  
  case 1: {  
    console.log('나머지는 1입니다.')  
  }  
  default: {  
    console.log('default 부분이 실행되었습니다.')  
  }  
}
```



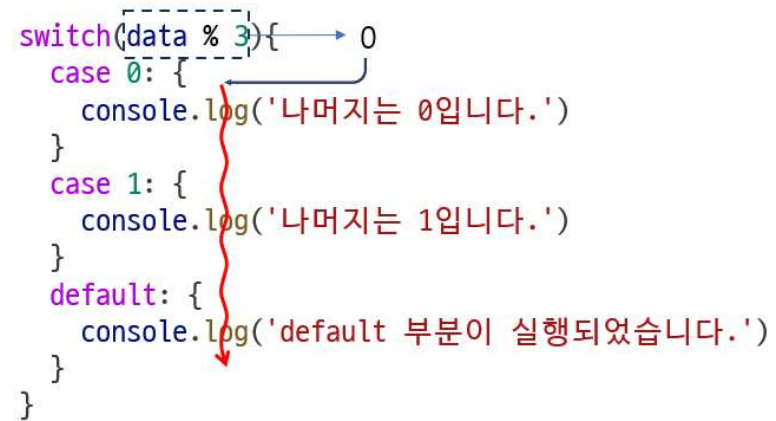
```
switch(data % 3){  
  case 0: {  
    console.log('나머지는 0입니다.')  
  }  
  case 1: {  
    console.log('나머지는 1입니다.')  
  }  
  default: {  
    console.log('default 부분이 실행되었습니다.')  
  }  
}
```



## switch - case

---

```
switch(data % 3){  
  case 0: {  
    console.log('나머지는 0입니다.')  
  }  
  case 1: {  
    console.log('나머지는 1입니다.')  
  }  
  default: {  
    console.log('default 부분이 실행되었습니다.')  
  }  
}
```



## switch - case

- 만약 특정 위치의 case 값에 일치하는 경우 그 위치의 case 만 실행되게 하고자 한다면 break 를 사용해 주어야 합니다.
- switch - case 에서 break 가 사용되었다면 break 에 의해 switch 영역을 벗어나게 됩니다.
- 즉 switch 부분의 실행이 끝나게 됨으로 break 가 작성된 하위 case 혹은 default 부분은 실행되지 않게 됩니다.

```
switch(data % 3){  
  case 0: {  
    console.log('나머지는 0입니다.')  
    break;  
  }  
  case 1: {  
    console.log('나머지는 1입니다.')  
    break;  
  }  
  default: {  
    console.log('default 부분이 실행되었습니다.')  
  }  
}
```



switch 부분 실행 종료



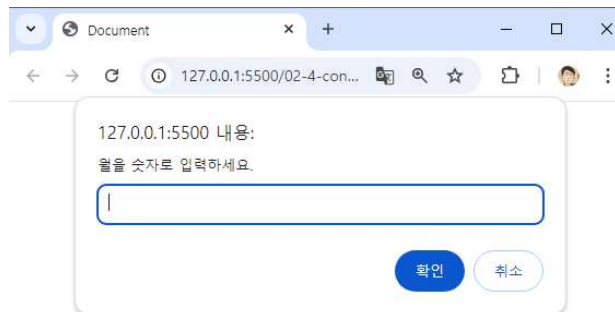
조건문이 여러 개 있는데 어떤 경우에 어떤 것을 사용하는 것이 좋은가요?



## switch 테스트

다음의 조건대로 코드를 작성하세요!!

- 공유된 파일의 switchTest() 함수 부분을 ifTest() 함수와 동일하게 동작하도록 switch-case 문을 이용해 작성한다.



# for

---

- 반복문은 특정 코드 부분을 반복적으로 실행시키기 위해 사용됩니다.
- 반복문은 for, while 문이 주로 사용되며 드물게 do – while 문이 사용되기도 합니다.
- for 문을 작성하는 방법은 특정 변수 값을 증감시켜 조건에 만족하는 동안 반복적으로 실행되게 하는 방법입니다.

```
for( ①초깃값 ; ②반복조건 ; ③증감식 ){  
  ④body, 반복적으로 실행될 구문  
}
```

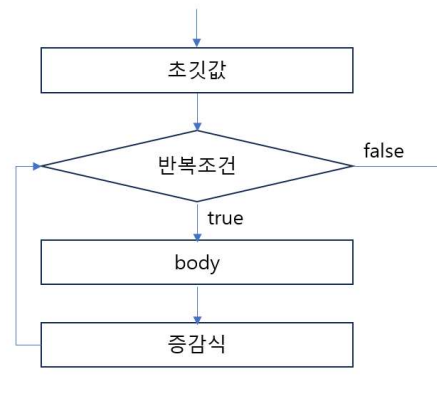
- 1 - 초기값 : 가장 처음 한번만 실행, for 반복을 위해 사용할 변수 초기화에 이용
- 2 - 반복조건 : 가장 처음 한번 실행, 이후 증감식이 실행된 후에 실행, body 부분이 실행될 것인지 판단
- 3 - 증감식 : body 가 실행된 후에 실행, 데이터를 증감시키기 위해 이용
- 4 – body : 반복조건이 만족하는 경우 계속 실행될 구문



# for

---

- for 문의 실행 흐름을 그림으로 그려보면 아래와 같습니다.



# for

---

```
for(let i = 0; i<3; i++){  
  console.log(`Hello ${i}`)  
}  
//Hello 0  
//Hello 1  
//Hello 2
```

반복 횟수	실행 순서	i 값
1	i = 0 → i<3 → console.log()	0
2	i++ → i<3 → console.log()	1
3	i++ → i<3 → console.log()	2
4	i++ → i<3 → 조건에 만족하지 않아 for 문 종료	3

# for

---

- for 문을 이용하면서 초깃값 부분에서 변수를 1개 이상 선언할 수도 있으며 증감식에서 값 증가 뿐만 아니라 감소를 명시할 수도 있습니다.

for 반복 테스트

```
1  for(let data1 = 1, data2 = 10; data1<=5 && data2>5; data1++, data2 -=  
2  2){
```



## 다음의 조건대로 코드를 작성하세요!!

- 공유된 파일을 이용해 아래처럼 구구단이 출력되게 for 문을 이용해 작성한다.
- 화면 출력은 document.write() 를 이용하세요.

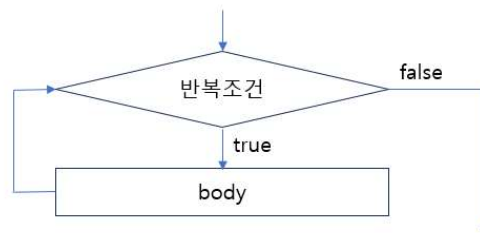
1단	2단	3단	4단	5단	6단	7단	8단	9단
1X1=1	2X1=2	3X1=3	4X1=4	5X1=5	6X1=6	7X1=7	8X1=8	9X1=9
1X2=2	2X2=4	3X2=6	4X2=8	5X2=10	6X2=12	7X2=14	8X2=16	9X2=18
1X3=3	2X3=6	3X3=9	4X3=12	5X3=15	6X3=18	7X3=21	8X3=24	9X3=27
1X4=4	2X4=8	3X4=12	4X4=16	5X4=20	6X4=24	7X4=28	8X4=32	9X4=36
1X5=5	2X5=10	3X5=15	4X5=20	5X5=25	6X5=30	7X5=35	8X5=40	9X5=45
1X6=6	2X6=12	3X6=18	4X6=24	5X6=30	6X6=36	7X6=42	8X6=48	9X6=54
1X7=7	2X7=14	3X7=21	4X7=28	5X7=35	6X7=42	7X7=49	8X7=56	9X7=63
1X8=8	2X8=16	3X8=24	4X8=32	5X8=40	6X8=48	7X8=56	8X8=64	9X8=72
1X9=9	2X9=18	3X9=27	4X9=36	5X9=45	6X9=54	7X9=63	8X9=72	9X9=81

# while

---

- while 문은 증감조건만 명시합니다.

```
while( 반복조건 ) {  
    body - 반복적으로 실행되는 구문  
}
```

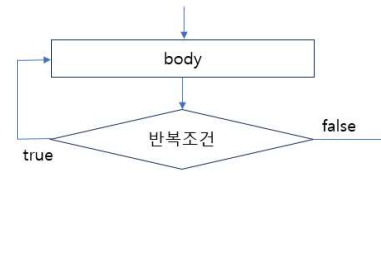


# do-while

---

- do – while 문도 while 문의 일종입니다.
- 반복 조건을 명시하고 그 조건이 true 이면 반복적으로 body 부분이 실행되는 구조입니다.
- do – while 문 마지막 부분에 조건을 명시하게 됩니다.
- 그럼으로 조건에 만족한다면 body 부분이 반복적으로 실행되는 것은 동일하지만 body 부분이 최초에 한번 실행이 되는지의 차이가 있습니다.

```
do {  
    body - 반복적으로 실행되는 구문  
} while( 반복조건 )
```





반복문을 작성할 때 `for`, `while` 중 어느 것을 이용하는 것이 좋은가요?



## while 테스트

다음의 조건대로 코드를 작성하세요!!

- 공유된 파일을 이용해 아래처럼 구구단이 출력되게 while 문을 이용해 작성한다.
- 화면 출력은 document.write() 를 이용하세요.

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5500/02-4-control-flow/test7-quiz-for/index.html'. The page content is a multiplication table with 9 columns and 9 rows. Each column is headed by a number from 1 to 9, and each row contains the products of that column header with row numbers 1 through 9.

1단	2단	3단	4단	5단	6단	7단	8단	9단
1X1=1	2X1=2	3X1=3	4X1=4	5X1=5	6X1=6	7X1=7	8X1=8	9X1=9
1X2=2	2X2=4	3X2=6	4X2=8	5X2=10	6X2=12	7X2=14	8X2=16	9X2=18
1X3=3	2X3=6	3X3=9	4X3=12	5X3=15	6X3=18	7X3=21	8X3=24	9X3=27
1X4=4	2X4=8	3X4=12	4X4=16	5X4=20	6X4=24	7X4=28	8X4=32	9X4=36
1X5=5	2X5=10	3X5=15	4X5=20	5X5=25	6X5=30	7X5=35	8X5=40	9X5=45
1X6=6	2X6=12	3X6=18	4X6=24	5X6=30	6X6=36	7X6=42	8X6=48	9X6=54
1X7=7	2X7=14	3X7=21	4X7=28	5X7=35	6X7=42	7X7=49	8X7=56	9X7=63
1X8=8	2X8=16	3X8=24	4X8=32	5X8=40	6X8=48	7X8=56	8X8=64	9X8=72
1X9=9	2X9=18	3X9=27	4X9=36	5X9=45	6X9=54	7X9=63	8X9=72	9X9=81



## break, continue

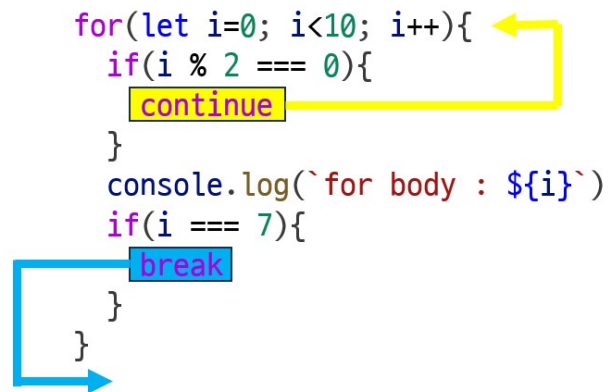
---

- for문 혹은 while, do – while 문을 이용해서 반복문을 작성하다 보면 break 혹은 continue 를 이용하는 경우가 있습니다.
- break 와 continue 는 반복문 내에 작성되어 반복문의 실행 흐름을 제어하기 위해서 사용됩니다.
- 물론 break 는 switch – case 문에도 사용되며 switch 문에 break 를 사용하면 switch 를 벗어나게 되어 제어하는 역할로도 사용됩니다.
- body 내에서 반복을 끝내야 하는 경우 혹은 반복 조건을 다시 판단해야 하는 경우가 있습니다.
- 이를 위해 제공되는 것이 break, continue 입니다.
- continue 와 break 는 차이가 있는데 반복문이 실행되다가 continue 를 만나게 되면 continue 아랫 부분은 실행되지 않으며 다시 반복 조건을 판단하게 됩니다.
- 그런데 break 문은 break 를 만나게 되면 반복문을 끝내게 됩니다.

# break, continue

---

```
for(let i=0; i<10; i++){  
  if(i % 2 === 0){  
    continue  
  }  
  console.log(`for body : ${i}`)  
  if(i === 7){  
    break  
  }  
}
```

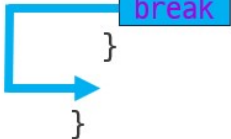


## break, continue

---

- break 와 continue 에 의해 제어되는 반복문은 break 와 continue 를 감싸고 있는 가장 가까운 반복문입니다.
- 이 부분이 중요할 수 있는데 이유는 반복문이 중복 작성 될 수 있기 때문입니다.

```
for(let no1=0; no1<2; no1++){  
  console.log(`position 1 : ${no1}`)  
  for(let no2=0; no2<2; no2++){  
    console.log(`position 2 : ${no1}, ${no2}`)  
    break  
  }  
}
```

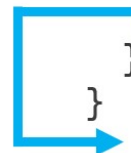


## break, continue

---

- 특정 위치의 반복문이 제어되게 하고 싶다면 라벨을 이용해야 합니다.
- 라벨이란 개발자가 지정하는 임의 식별자이며 반복문에 식별자를 선언하고 break 에서 그 라벨을 명시하여 해당 반복문이 제어되게 할 수 있습니다.

```
myloop : for(let no1=0; no1<2; no1++){  
  console.log(`position 1 : ${no1}`)  
  for(let no2=0; no2<2; no2++){  
    console.log(`position 2 : ${no1}, ${no2}`)  
    break myloop  
  }  
}
```

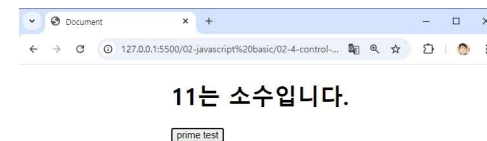




## control flow 종합 테스트 #1

다음의 조건대로 코드를 작성하세요!!

- 공유된 파일을 이용해 숫자를 사용자에게 입력 받고 그 입력한 숫자가 소수인지를 판단하는 프로그램을 작성한다.
- 소수는 1보다 큰 자연수중 1과 자신으로만 나누어지는 수를 의미한다.

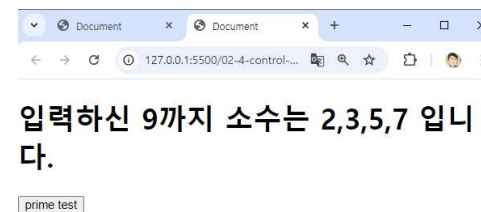
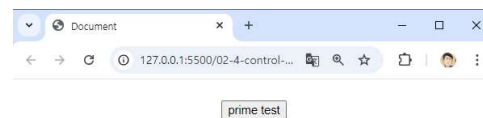




## control flow 종합 테스트 #2

다음의 조건대로 코드를 작성하세요!!

- 공유된 파일을 이용해 숫자를 사용자에게 입력 받고 1부터 입력한 숫자까지 모든 소수를 출력하는 프로그램을 작성
- 소수는 1보다 큰 자연수중 1과 자신으로만 나누어지는 수를 의미한다.





# 감사합니다

단단히 마음먹고 떠난 사람은  
산꼭대기에 도착할 수 있다.  
산은 올라가는 사람에게만 정복된다.



윌리엄 셰익스피어  
William Shakespeare