JavaScript

제어문 – 김근형 강사

제어문

- JavaScript 제어문
  - 제어문이란 로직을 제어할 때 쓰이는 명령문의 모음이다
  - 크게는 조건문과 반복문으로 나뉘게 되며 javascript에서도 조건문과 반복문이 존재한다.
  - 조건문에는 javascript에서 if와 switch~case 문이 있다
  - 반복문은 javascript에서 while과 for문이 있다.

조건문 - if

- 조건문 – if
  - 단순 조건 if문 : 해당 조건식이 참일 경우에만 로직을 실행하고 그렇지 못할 경우 로직을 실행시키지 않고 if 문 바로 다음의 문장을 실행시킨다.

```
if(조건문){
    조건이 참일 시 사용할 로직
}
```

```
let s = 1;
if(s>0){
    console.log('해당 숫자는 양수입니다.');
}
```

```
해당 숫자는 양수입니다.
```

조건문 - if

- 조건문 – if
  - 단순 조건 if문 : 해당 조건식이 참일 경우에만 로직을 실행하고 그렇지 못할 경우 로직을 실행시키지 않고 if 문 바로 다음의 문장을 실행시킨다.

```
if(조건문){
    조건이 참일 시 사용할 로직
}
조건에 상관없이 실행되는 로직
```

```
var s = 1;
if(s>0){
    console.log('해당 숫자는 양수입니다.');
}
console.log("이 로직은 if문에 상관없이 실행됩니다");
```

```
해당 숫자는 양수입니다.
이 로직은 if문에 상관없이 실행됩니다
```

조건문 - if

- 조건문 – if ~ else
  - if ~ else 문 : 해당 조건식이 참일 경우 if문의 로직을 실행하고 거짓일 경우 else의 로직을 실행한다.
  - 실행한 뒤에 if ~ else 문 아래의 로직을 실행시킨다.

```
if(조건문){
        조건이 참일 시 사용할 로직
}else {
        조건이 거짓일 시 사용할 로직
}
조건에 상관없이 실행되는 로직
```

조건문 - if

- 조건문 – if ~ else
  - if ~ else 문 : 해당 조건식이 참일 경우 if문의 로직을 실행하고 거짓일 경우 else의 로직을 실행한다.
  - 실행한 뒤에 if ~ else 문 아래의 로직을 실행시킨다.

```
var a = 3;
if(a < 2){
    console.log("a는 2보다 작습니다.");
}else{
    console.log("a는 2보다 큽니다.");
}
console.log("if~else문과는 상관없이 쓰는 로직");
```

```
a는 2보다 큽니다.
if~else문과는 상관없이 쓰는 로직
```

조건문 - if

- 조건문 – if ~ else if ~ else
  - if ~ else if ~ else 문 : if ~ else if ~ else 문은 해당 조건이 만족할 경우 해당 조건에 있는 로직을 실행시키고 if 전체 로직을 종료하며 바로 밑에 로직을 실행시킨다.
  - else if문은 중첩해서 사용이 가능하다.
  - if ~ else if ~ else 문의 조건에 부합하는 조건들이 없다면 else 문의 조건을 수행한다.
  - 단 if ~ else if ~ else 문의 else는 생략이 가능하다.

```
if(조건문){
        해당 조건이 참일 시 사용할 로직
} else if(조건문){
        해당 조건이 참일 시 사용할 로직
} else if(조건문){
        해당 조건이 참일 시 사용할 로직
} else if(조건문){
        ……
} else {
        모든 조건을 만족하지 못했을 시 사용하는 로직
}
조건에 상관없이 실행되는 로직
```

조건문 - if

- 조건문 – if ~ else if ~ else
  - if ~ else if ~ else 문 : if ~ else if ~ else 문은 해당 조건이 만족할 경우 해당 조건에 있는 로직을 실행시키고 if 전체 로직을 종료하며 바로 밑에 로직을 실행시킨다.
  - else if문은 중첩해서 사용이 가능하다.
  - if ~ else if ~ else 문의 조건에 부합하는 조건들이 없다면 else 문의 조건을 수행한다.
  - 단 if ~ else if ~ else 문의 else는 생략이 가능하다.

```
var a = 3;
if(a == 1){
    console.log("a는 1입니다.");
}else if(a == 2){
    console.log("a는 2입니다.");
}else if(a == 3){
    console.log("a는 3입니다.");
}else if(a == 4){
    console.log("a는 4입니다.");
}else {
    console.log("a에 해당되는 값이 없습니다.");
}
console.log("if ~ else if ~ else 문과는 상관없이 쓰는 로직");
```

```
a는 3입니다.
if ~ else if ~ else 문과는 상관없이 쓰는 로직
```

조건문 - if

- 조건문 – 중첩 if 문
  - 중첩 if문 : if문 안에 if문을 넣어서 실행시키는 문장을 의미한다.
  - if문 안에 if문을 몇 개를 넣든 제한은 없다.
  - if ~ else if ~ else 문장 안에서 if문 또한 선언이 가능하며 어디서든 if문의 중복 선언이 가능하다.

```
if(조건문){
        if(조건문){
                if(조건문){
                        …
                }
        }
} else if(조건문){
        if(조건문){
                …
        }
} else {
        if(조건문){
                …
        }
}
```

조건문 - if

- 조건문 – 중첩 if 문

```
var a = 5;
if(a>3){
    if (a==4) {
        console.log("a는 4입니다.");
    }else if(a==5){
        console.log("a는 5입니다.");
    }else {
        console.log("a는 해당하는 값이 없습니다.");
    }
}else{
    if(a==1){
        console.log("a는 1입니다.");
    }else if(a == 2){
        console.log("a는 2입니다.");
    }else{
        console.log("a는 해당하는 값이 없습니다.");
    }
}
```

조건문 – switch ~ case

- 조건문 – switch ~ case
  - javascript의 조건문의 또 다른 형태
  - switch ~ case는 if문을 다르게 표현할 경우 사용한다.
  - if문과의 차이점은 if문은 비교식을 통한 참, 거짓을 이용해 분기를 태우지만 switch문은 오로지 동일한 값의 비교를 통한 동등 비교만이 가능하다.
  - switch문에 들어가는 조건값은 비교하고자 하는 변수 혹은 값이 되고 case 문의 비교값은 해당 변수의 비교 대상이다.
  - case문의 비교값이 조건값과 같을 경우 해당 로직을 실행시킨다.
  - 만약 case문의 비교값과 조건값이 전부 불일치하면 default의 로직을 실행시킨다.

```
switch(조건값){
case 비교값1 :
      실행시키고자 하는 로직;
      break;
case 비교값2 :
      실행시키고자 하는 로직;
      break;
case 비교값3 :
      ……
default :
      실행시키고자 하는 로직;
}
```

조건문 – switch ~ case

- 조건문 – switch ~ case

```
var a = 2;
switch(a){
case 1 :
    console.log("a는 1입니다.");
    break;
case 2 :
    console.log("a는 2입니다.");
    break;
case 3 :
    console.log("a는 3입니다.");
    break;
default :
    console.log("a에 존재하는 값이 없습니다.");
}
```

```
a는 2입니다.
```

조건문 – switch ~ case

- 조건문 – switch ~ case
  - 기존의 if ~ else if ~ else 문은 한 분기를 태우면 if문의 밖으로 나가지만 switch~case문은 한 분기를 태우게 되면 아래의 분기를 모두 실행하게 된다.
  - 그렇기에 switch~case문의 모든 분기 아래에는 break문을 써서 분기를 강제로 태우지 못하도록 만든다.

```
var a = 2;
switch(a){
case 1 :
    console.log("a는 1입니다.");
case 2 :
    console.log("a는 2입니다.");
case 3 :
    console.log("a는 3입니다.");
default :
    console.log("a에 존재하는 값이 없습니다.");
}
```

```
a는 2입니다.
a는 3입니다.
a에 존재하는 값이 없습니다.
```

조건문 – switch ~ case

- 조건문 – switch ~ case
  - 이러한 특징을 역으로 이용한다면 범위를 이용한 분기 작성이 가능하다.

```
var a = 2;
switch(a){
case 1 :
case 2 :
case 3 :
case 4 :
case 5 :
case 6 :
case 7 :
    console.log("a는 1부터 7 사이의 숫자입니다.");
    break;
case 8 :
case 9 :
case 10 :
    console.log("a는 8부터 10 사이의 숫자입니다.");
    break;
default :
    console.log("a에 존재하는 값이 없습니다.");
}
```

```
a는 1부터 7 사이의 숫자입니다.
```

반복문 – while

- 반복문 - while
  - while 문의 조건식이 참일 동안에 로직을 실행시킨다.
  - while 문의 조건식이 거짓일 경우 로직을 빠져나와 바로 아래 로직을 실행시킨다.
  - 로직을 잘못 설계한 경우 무한 루프를 돌 수 있으므로 주의해야 한다.

```
while(조건문){
    조건이 참일 시 반복할 로직
}
```

```
var a = 0;
while(a<5){
    console.log(a);
    a++;
}
```

```
0
1
2
3
4
```

반복문 - while

- 반복문 - do ~ while
  - while문은 맨 처음의 조건식이 false일 경우 아예 로직을 태우지를 않는다.
  - 반드시 한번의 로직을 반복문에서 태우기 위해서는 do~while문을 쓴다.
  - do~while문은 로직을 어떤 경우든 반드시 한번을 태우고 반복의 조건식을 비교하는 점이 다르다.

```
do{
    조건이 참일 시 반복할 로직
} while(조건문);
```

```
var a = 0;
do{
    console.log(a);
    a++;
}while(a<5);
```

반복문 - while

- continue
  - 반복문에서 로직을 스킵하려고 할 경우 사용하는 명령어
  - continue 문장을 사용하게 되면 continue문 아래의 로직을 무시하고 다시 반복문의 비교식으로 돌아와 조건을 비교한다.
  - 만약 while문 아래에 증감연산이 있을 경우 증감연산 자체가 무시될 수 있으므로 위치 선정에 주의하여야 한다.

```
var a = 0;
while(a<5){
    a++;
    // if문 안의 로직이 하나만 있을 경우 아래와 같이 사용 가능하다.
    if(a==3)continue;
    console.log(a);
}
```

```
1
2
4
5
```

반복문 - while

- continue
  - 증감연산의 위치가 continue 문 보다 아래 있게 되면 일정 조건이 되었을 경우 continue문을 태우지 못한다.
  - 즉 다시 이야기해서 continue문 아래에는 증감연산을 놓지 않도록 하는 것이 가장 좋다.
  - 이런 위치 선정 때문에 while문 안에서 continue문을 사용할 경우 종종 로직이 바뀌는 경우가 생긴다.

```
var a = 0
// 이 로직은 무한 루프를 탄다.
while(a < 5){
    console.log(a);
    if(a==3)continue;
    a++;
}
```

```
0
1
2
1398 3
```

반복문 - while

- break;
  - break문은 continue와 같이 아래의 로직을 생략하는 것은 동일하다.
  - 단 continue문은 반복문의 맨 처음 조건식으로 돌아가는 반면 break문은 해당 while문 로직을 아예 빠져나와 버린다.
  - 조건식 만으로 루프를 빠져나오는 구문을 만들기 힘들 경우 break문을 통해 빠져나오게 할 수 있다.

```
var a = 0;
while(a<5){
    console.log(a);
    if(a==3)break;
    a++
}
```

```
0
1
2
3
```

반복문 - for

- 반복문 – for
  - 기존의 while문은 카운터 변수와 증감연산의 관리가 쉽지 않았다.
  - 기존의 ()를 확장하여 초기값, 비교식, 증감연산을 한꺼번에 할 수 있는 제어문이 for문이다.
  - for문에서는 초기값과 비교식, 증감연산을 같이 제어할 수 있기 때문에 상당히 유용한 제어문이다.

```
for(초기값;비교식;증감연산){
    조건이 참일 시 반복할 로직
}
```

```
for (var i = 0; i < 5; i++) {
    console.log(i);
}
```

```
0
1
2
3
4
```

반복문 - for

- 반복문 – for
  - 반복문의 초기변수와 조건식 증감 값은 각각 생략이 가능하며 만약 조건식을 생략할 경우 무조건 true라는 뜻이므로 무한 루프를 돌게 된다.
  - for문 또한 continue 와 break 문을 쓸 수가 있으며 위와 같은 무한 루프를 돌 수가 있다.

```
var a = 0;
for (; ;) {
    console.log(a);
    if(a==5)break;
    a++;
}
```

```
0
1
2
3
4
5
```

다중 반복문

- 중첩 반복문
  - 반복문 안에 반복문을 넣어 활용할 수 있다.
  - 반복문이 두 번 이상 중첩이 되면 반복문에서 돌아가는 로직에 대한 파악이 쉽지 않다.
  - 각 루프마다 초기값의 변화와 로직의 실행 결과를 직접 하나하나 적어보며 로직을 따라가야 한다.

```
for (var i = 0; i < 3 ; i++) {
    for (var j = 0; j < 3; j++) {
        console.log(i+" "+j);
    }
}
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

label

- label 문
  - break나 continue를 사용하게 되면 현재 속해있는 바로 위의 반복문을 빠져나오거나 해당 반복문의 처음으로 다시 돌아온다.
  - 하지만 중첩 반복문의 경우 가장 안에 있는 for문 안에서 break나 continue를 넣었을 때 상위의 반복문을 빠져 나온다거나 아니면 최상위 반복문의 처음으로 돌아가는 동작은 할 수가 없다.
  - 라벨이라고 하는 문장은 break나 continue문을 실행시켰을 시 돌아갈 수 있는 지표를 찍어주는 문장이다.
  - 혹은 라벨을 붙인 블록을 빠져나갈 경우 사용하기도 한다.
  - 라벨을 쓰기 위해서는 for, while 혹은 브레이스({}) 앞에 '임의의 이름 :' 을 붙여준다.
  - 라벨은 잘못 쓸 경우 디버깅이나 코드의 가독성을 떨어트릴 수 있으니 주의해서 사용하도록 한다.

label

- label – continue

```
var i, j;

loop1:
for (i = 0; i < 3; i++) {      //첫번째 for문은 "loop1" 레이블을 붙였다.
   loop2:
   for (j = 0; j < 3; j++) {   //두번째 for문은 "loop2" 레이블을 붙였다.
      if (i === 1 && j === 1) {
         continue loop1;
      }
      console.log('i = ' + i + ', j = ' + j);
   }
}
```

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 1, j = 0
i = 2, j = 0
i = 2, j = 1
i = 2, j = 2
```

label

- label – break

```
var i, j;

loop1:
for (i = 0; i < 3; i++) {      //The first for statement is labeled "loop1"
   loop2:
   for (j = 0; j < 3; j++) {   //The second for statement is labeled "loop2"
      if (i === 1 && j === 1) {
         break loop1;
      }
      console.log('i = ' + i + ', j = ' + j);
   }
}
```

```
i = 0, j = 0
i = 0, j = 1
i = 0, j = 2
i = 1, j = 0
```

label

- label – block

```
foo: {
  console.log('face');
  break foo;
  console.log('this will not be executed');
}
console.log('swap');
```

```
face
swap
```