# 221207_2반_실습

# JavaScript

## 반복문

### for 문

- for 반복문은 어떤 특정한 조건이 거짓으로 판별될 때까지 반복합니다. 자바스크립트의 반복문은 C의 반복문과 비슷합니다. for 반복문은 다음과 같습니다.

```
for ([초기문]; [조건문]; [증감문]) {
      문장
}
```

- for문이 실행될 때, 다음과 같이 실행됩니다.
  1. 초기화 구문인 초기문이 존재한다면 초기문이 실행됩니다. 이 표현은 보통 1이나 반복문 카운터로 초기 설정이 됩니다. 그러나 복잡한 구문으로 표현 될 때도 있습니다. 또한 변수로 선언 되기도 합니다
  2. 조건문은 조건을 검사합니다. 만약 조건문이 참이라면, 그 반복문은 실행됩니다. 만약 조건문이 거짓이라면, 그 for문은 종결됩니다. 만약 그 조건문이 생략된다면, 그 조건문은 참으로 추정됩니다.
  3. 문장이 실행됩니다. 많은 문장을 실행할 경우엔, { } 를 써서 문장들을 묶어 줍니다.
  4. 갱신 구문인 증감문이 존재한다면 실행되고 2번째 단계로 돌아갑니다.
- 예시

```
<form name="selectForm">
  <p>
    <label for="musicTypes">Choose some music types, then click the button below:
</label>
    <select id="musicTypes" name="musicTypes" multiple="multiple">
      <option selected="selected">R&B</option>
      <option>Jazz</option>
      <option>Blues</option>
      <option>New Age</option>
      <option>Classical</option>
      <option>Opera</option>
    </select>
  </p>
  <p><input id="btn" type="button" value="How many are selected?" /></p>
```

```
</form>

<script>
function howMany(selectObject) {
  var numberSelected = 0;
  for (var i = 0; i < selectObject.options.length; i++) {
    if (selectObject.options[i].selected) {
      numberSelected++;
    }
  }
  return numberSelected;
}

var btn = document.getElementById("btn");
btn.addEventListener("click", function(){
  alert('Number of options selected: ' + howMany(document.selectForm.musicTypes))
});
</script>
```

```
for (var i = 0; i < selectObject.options.length; i++) {
  if (selectObject.options[i].selected) {
    numberSelected++;
  }
}
```

## do… while 문

- do...while 문은 특정한 조건이 거짓으로 판별될 때까지 반복합니다. do...while 문은 다음과 같습니다.

```
do {
      문장
} while (조건문);
```

- **조건문을 확인하기 전에 문장은 한번 실행됩니다.** 많은 문장을 실행하기 위해선 { }를 써서 문장들을 묶어줍니다. 만약 조건이 참이라면, 그 문장은 다시 실행됩니다. 매 실행 마지막마다 조건문이 확인됩니다. 만약 조건문이 거짓일 경우, 실행을 멈추고 do...while 문 바로 아래에 있는 문장으로 넘어가게 합니다.
- 예시

```
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

## while 문

- while 문은 어떤 조건문이 참이기만 하면 문장을 계속해서 수행합니다. while 문은 다음과 같습니다.

```
while (조건문) {
      문장
}
```

- 만약 조건문이 거짓이 된다면, 그 반복문 안의 문장은 실행을 멈추고 반복문 바로 다음의 문장으로 넘어갑니다.
- 조건문은 반복문 안의 문장이 실행되기 전에 확인 됩니다. 만약 조건문이 참으로 리턴된다면, 문장은 실행되고 그 조건문은 다시 판별됩니다. 만약 조건문이 거짓으로 리턴된다면, 실행을 멈추고 while문 바로 다음의 문장으로 넘어가게 됩니다.
- 많은 문장들을 실행하기 위해선, { }를 써서 문장들을 묶어줍니다.
- 예시 1

```
n = 0;
x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

  - 매 반복과 함께, n이 증가하고 x에 더해집니다. 그러므로, x와 n은 다음과 같은 값을 갖습니다.
    - 첫번째 경과 후: `n` = 1 and `x` = 1
    - 두번째 경과 후: `n` = 2 and `x` = 3
    - 세번째 경과 후: `n` = 3 and `x` = 6
  - 세번째 경과 후에, n < 3 은 더이상 참이 아니므로, 반복문은 종결됩니다.
- 예시 2

```
// 다음과 같은 코드는 피하세요.
while (true) {
  console.log("Hello, world");
}
```

## Label 문

- 여러분이 프로그램에서 다른 곳으로 참조할 수 있도록 식별자로 문을 제공합니다. 예를 들어, 여러분은 루프를 식별하기 위해 레이블을 사용하고, 프로그램이 루프를 방해하거나 실행을 계속할지 여부를 나타내기 위해 break나 continue 문을 사용할 수 있습니다.

```
label :
       statement
```

- 레이블 값은 예약어가 아닌 임의의 JavaScript 식별자일 수 있습니다. 여러분이 레이블을 가지고 식별하는 문은 어떠한 문이 될 수 있습니다.
- 예시

```
markLoop:
while (theMark == true) {
   doSomething();
}
//레이블 markLoop는 while 루프를 식별합니다.
```

## break 문

- break문은 반복문, switch문, 레이블 문과 결합한 문장을 **빠져나올 때 사용**합니다.
  - 레이블 없이 break문을 쓸 때: 가장 가까운 `while`, `do-while`, `for`, 또는 `switch`문을 종료하고 다음 명령어로 넘어갑니다.
  - 레이블 문을 쓸 때: 특정 레이블 문에서 끝납니다.
- break문의 문법은 다음과 같습니다.
  1. `break;`
  2. `break [레이블];`
- break문의 첫번째 형식은 가장 안쪽의 반복문이나 switch문을 빠져나옵니다. 두번째 형식는 특정한 레이블 문을 빠져나옵니다.
- 예시1

```
for (i = 0; i < a.length; i++) {
  if (a[i] == theValue) {
    break;
  }
}
```

- 예시 2 : Breaking to a label

```
var x = 0;
var z = 0
labelCancelLoops: while (true) {
  console.log("Outer loops: " + x);
  x += 1;
  z = 1;
  while (true) {
    console.log("Inner loops: " + z);
    z += 1;
    if (z === 10 && x === 10) {
      break labelCancelLoops;
    } else if (z === 10) {
      break;
    }
  }
}
```

## Continue 문

- `continue` 문은 while, do-while, for, 레이블 문을 다시 시작하기 위해 사용될 수 있습니다.
  - 레이블없이 continue를 사용하는 경우, 그것은 가장 안쪽의 while, do-while, for 문을 둘러싼 현재 반복을 종료하고, 다음 반복으로 루프의 실행을 계속합니다. break 문과 달리, continue 문은 전체 루프의 실행을 종료하지 않습니다. while 루프에서 그것은 다시 조건으로 이동합니다. for 루프에서 그것은 증가 표현으로 이동합니다.
  - 레이블과 함께 continue를 사용하는 경우, continue는 그 레이블로 식별되는 루프 문에 적용됩니다.
- continue 문의 구문은 다음과 같습니다:
  1. `continue;`
  2. `continue label;`
- 예시 1

```
i = 0;
n = 0;
while (i < 5) {
  i++;
  if (i == 3) {
    continue;
  }
  n += i;
}
```

```
//i 값이 3일 때 실행하는 continue 문과 함께 while 루프를 보여줍니다. 따라서, n은 값 1, 3, 7, 1
2를 취합니다.
```

- 예시 2

```
checkiandj:
  while (i < 4) {
    console.log(i);
    i += 1;
    checkj:
      while (j > 4) {
        console.log(j);
        j -= 1;
        if ((j % 2) == 0) {
          continue checkj;
        }
        console.log(j + " is odd.");
      }
      console.log("i = " + i);
      console.log("j = " + j);
  }
```

  - checkiandj 레이블 문은 checkj 레이블 문을 포함합니다. continue가 발생하는 경우, 프로그램은 checkj의 현재 반복을 종료하고, 다음 반복을 시작합니다. 그 조건이 false를 반환 할 때까지 continue가 발생할 때마다, checkj는 반복합니다. false가 반환될 때, checkiandj 문의 나머지 부분은 완료되고, 그 조건이 false를 반환 할 때까지 checkiandj는 반복합니다. false가 반환될 때, 이 프로그램은 다음 checkiandj 문에서 계속됩니다.
  - continue가 checkiandj의 레이블을 가지고 있다면, 프로그램은 checkiandj 문 상단에서 계속될 것입니다.

## for… in 문

- for… in 문은 객체의 열거 속성을 통해 지정된 변수를 반복합니다. 각각의 고유한 속성에 대해, JavaScript는 지정된 문을 실행합니다. for...in 문은 다음과 같습니다:

```
for (variable in object) {
  statements
}
```

- 예시

```
//다음 함수는 객체와 객체의 이름을 함수의 인수로 취합니다. 그런 다음 모든 객체의 속성을 반복하고 속성
 이름과 값을 나열하는 문자열을 반환합니다.
```

```
function dump_props(obj, obj_name) {
  var result = "";
  for (var i in obj) {
    result += obj_name + "." + i + " = " + obj[i] + "<br>";
  }
  result += "<hr>";
  return result;
}
```

- 속성 make와 model을 가진 객체 car의 경우, 결과는 다음과 같습니다:

```
car.make = Ford
car.model = Mustang
```

- 배열
  - `배열`요소를 반복하는 방법으로 이를 사용하도록 유도될 수 있지만, **for...in** 문은 숫자 인덱스에 추가하여 사용자 정의 속성의 이름을 반환합니다. 따라서 만약 여러분이 사용자 정의 속성 또는 메서드를 추가하는 등 Array 객체를 수정한다면, 배열 요소 이외에도 사용자 정의 속성을 통해 **for...in** 문을 반복하기 때문에, 배열을 통해 반복할 때 숫자 인덱스와 전통적인 `for` 루프를 사용하는 것이 좋습니다.

## for… of 문

- `for...of` 문은 각각의 고유한 특성의 값을 실행할 명령과 함께 사용자 지정 반복 후크를 호출하여, 반복 가능한 객체(`배열`, `Map` (en-US), `Set`, 인수 객체 등을 포함)를 통해 반복하는 루프를 만듭니다.

```
for (variable of object) {
      statement
    }
```

- 다음 예는 for...of 루프와 `for...in` 루프의 차이를 보여줍니다. 속성 이름을 통해 for...in이 반복하는 동안, for...of은 속성 값을 통해 반복합니다:

```
let arr = [3, 5, 7];
arr.foo = "hello";

for (let i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (let i of arr) {
   console.log(i); // logs "3", "5", "7"
}
```

# 조건문

## if … else 문

- 기본 if … else 문법

```
if (조건) {
  만약 조건(condition)이 참일 경우 실행할 코드
} else {
  대신 실행할 다른 코드
}
```

```
if (조건) {
  만약 조건(condition)이 참일 경우 실행할 코드
}

실행할 다른 코드
```

- 예시

```
let shoppingDone = false;
let childsAllowance;

if (shoppingDone === true) {
  childsAllowance = 10;
} else {
  childsAllowance = 5;
}
```

## else if

- 두 가지보다 더 많은 것을 원할때 사용
- 예시

```
let year = prompt('ECMAScript-2015 명세는 몇 년도에 출판되었을까요?', '');

if (year < 2015) {
  alert( '숫자를 좀 더 올려보세요.' );
} else if (year > 2015) {
  alert( '숫자를 좀 더 내려보세요.' );
} else {
  alert( '정답입니다!' );
}
```

## 조건부 연산자 '?'

- 문법

```
let result = condition ? value1 : value2;
```

- 예시

```
let accessAllowed;
let age = prompt('나이를 입력해 주세요.', '');

if (age > 18) {
  accessAllowed = true;
} else {
  accessAllowed = false;
}

alert(accessAllowed);
```

  - '물음표(question mark) 연산자'라고도 불리는 '조건부(conditional) 연산자'를 사용하면 위 예시를 더 짧고 간결하게 변형할 수 있습니다.
  - 조건부 연산자는 물음표 `?`로 표시합니다. 피연산자가 세 개이기 때문에 조건부 연산자를 '삼항(ternary) 연산자'라고 부르는 사람도 있습니다. 참고로, 자바스크립트에서 피연산자를 3개나 받는 연산자는 조건부 연산자가 유일합니다.

```
let accessAllowed = (age > 18) ? true : false;
```

  - `age > 18` 주위의 괄호는 생략 가능합니다. 물음표 연산자는 우선순위가 낮으므로 비교 연산자 `>`가 실행되고 난 뒤에 실행됩니다.
  - 아래 예시는 위 예시와 동일하게 동작합니다.

```
// 연산자 우선순위 규칙에 따라, 비교 연산 'age > 18'이 먼저 실행됩니다.
// (조건문을 괄호로 감쌀 필요가 없습니다.)
let accessAllowed = age > 18 ? true : false;
```

  - 괄호가 있으나 없으나 차이는 없지만, 코드의 가독성 향상을 위해 괄호를 사용할 것을 권유합니다.

**주의:**

비교 연산자 자체가 `true`나 `false`를 반환하기 때문에 위 예시에서 물음표 연산자를 사용하지 않아도 됩니다.

```
// 동일하게 동작함
let accessAllowed = age > 18;
```