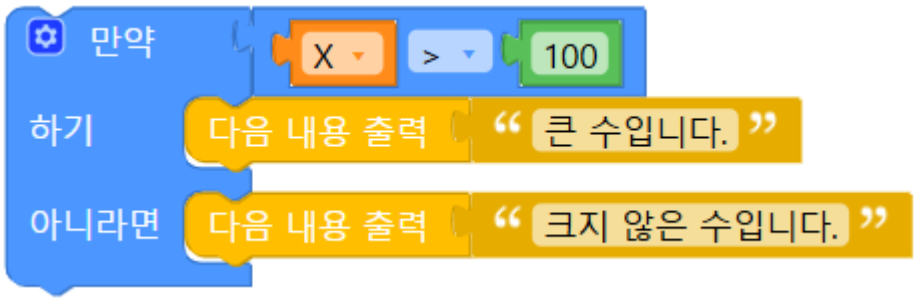
기본 블록

논리

논리 블록은 일반적으로 조건 블록과 반복 블록을 제어하는 데 사용됩니다.

다음은 예시입니다:



변수 **x** 의 값이 100 보다 크면 조건은 **참**이 되어“큰 수입니다.”라는 텍스트가 출력됩니다.
만약 **x** 의 값이 100 보다 크지 않으면 조건은 **거짓**이 되어“크지 않은 수입니다.”가 출력됩니다.

불리언 (boolean) 은 두 가지 값을 갖는 간단한 수학적 시스템입니다:
- **참**
- **거짓**

불리언 값은 변수에 저장하거나 함수로 전달할 수 있으며, 숫자, 텍스트, 리스트 값과 동일한 방식으로 사용할 수 있습니다.

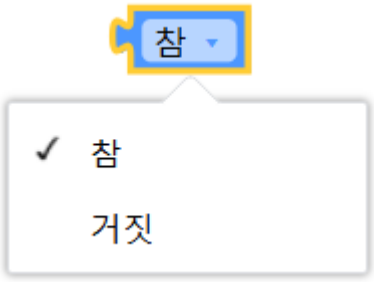
블록

블록이 불리언 값을 입력으로 요구할 때, 입력이 없으면 일반적으로 **거짓**으로 해석됩니다.

불리언 값이 기대되는 곳에 불리언이 아닌 값을 직접 연결할 수는 없지만,
그 값을 변수에 저장한 뒤 그것을 입력으로 전달하는 것은 가능합니다.
다만, 이는 권장되지 않으며, 향후 버전에서 동작이 달라질 수 있습니다.

값

참 또는 **거짓**를 지정하는 드롭다운이 있는 단일 블록을 사용하여 불리언 값을 얻을 수 있습니다:



Javascript 코드

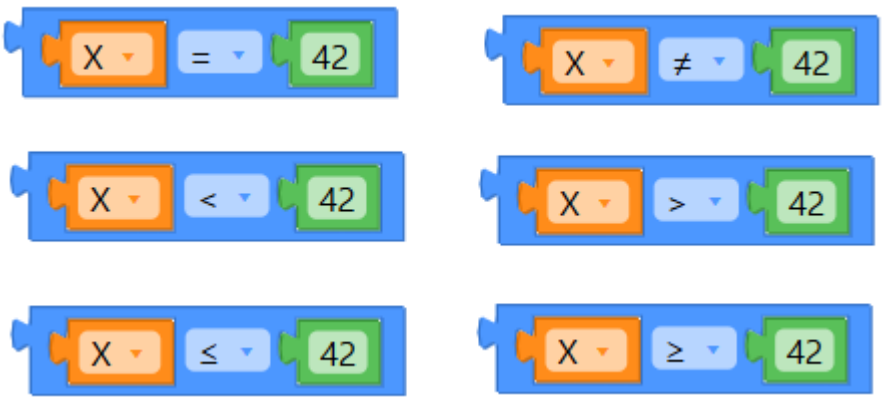
```
true    // 참
false   // 거짓
```

Python 코드

```
True    # 참
False   # 거짓
```

비교

여섯 가지 비교 연산자가 있습니다.
각 연산자는 두 개의 입력 (보통 숫자) 을 받아, 입력값들이 서로 어떻게 비교되는지에 따라 참 또는 거짓을 반환합니다.



여섯 가지 연산자는 다음과 같습니다:
- 같다 - 같지 않다 - 작다 - 크다 - 작거나 같다 - 크거나 같다

Javascript 코드

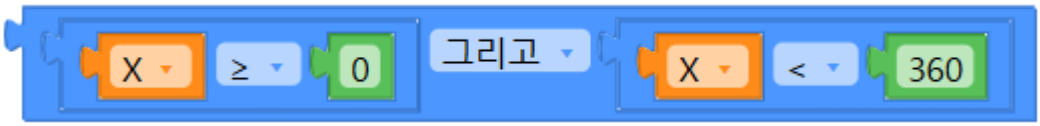
```
x == 42    // 같다
x != 42    // 같지 않다
x < 42     // 작다
x > 42     // 크다
x <= 42    // 작거나 같다
x >= 42    // 크거나 같다
```

Python 코드

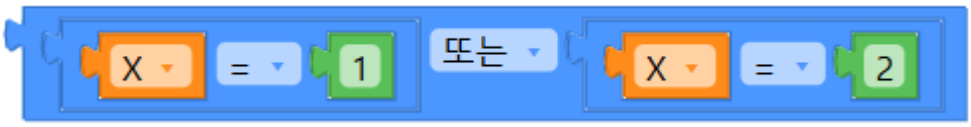
```
x == 42    # 같다
x != 42    # 같지 않다
x < 42     # 작다
x > 42     # 크다
x <= 42    # 작거나 같다
x >= 42    # 크거나 같다
```

논리 연산

그리고 블록은 두 입력이 모두 참일 때만 참을 반환합니다.



또는 블록은 두 입력 중 하나라도 참이면 참을 반환합니다.



Javascript 코드

```
x >= 0 && x < 360   // 그리고
x == 1 || x == 2  // 또는
```

Python 코드

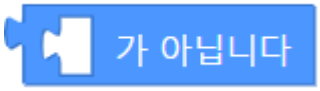
```
x >= 0 and x < 360  # 그리고
x == 1 or x == 2    # 또는
```

아닙니다

아닙니다 블록은 불리언 입력을 그 반대로 변환합니다. 예를 들어, 다음과 같은 결과는:

거짓이 됩니다.

위에서 언급한 바와 같이, 입력이 없으면 **참** 값이 기본값으로 간주되므로, 아래 블록은 **거짓** 값을 반환합니다:



입력을 비워두는 것은 권장되지 않습니다.

Javascript 코드

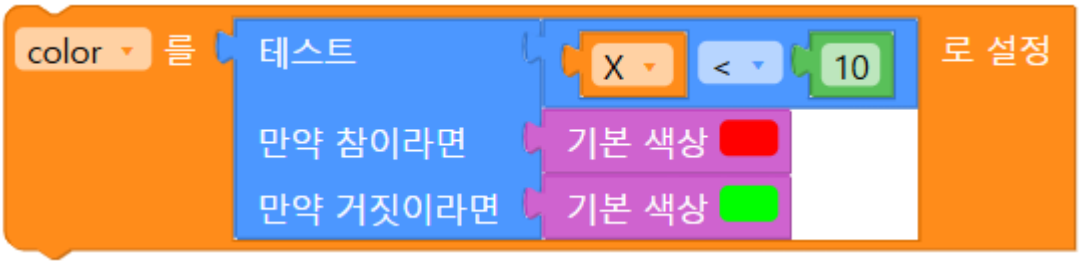
```
!true       // 또는 false
```

Python 코드

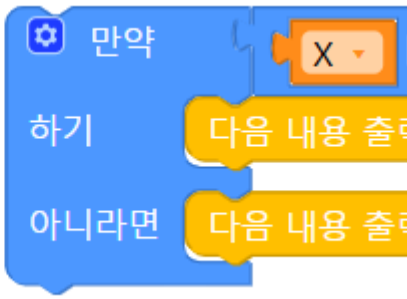
```
not True    # 또는 False
```

삼항 연산자 (Ternary operator)

삼항 블록은 간단한 조건문 블록처럼 동작합니다.
세 개의 입력을 받습니다.
첫 번째 입력은 테스트할 불리언 조건이고, 두 번째 입력은 조건이 **참**일 경우 반환할 값입니다. 세 번째 입력은 조건이 **거짓**일 경우 반환할 값입니다.
아래 예시에서 **x** 변수가 10 보다 작으면 **color** 변수는 빨간색으로 설정되고, 그렇지 않으면 **color** 변수는 초록색으로 설정됩니다.



삼항 블록은 항상 조건문 블록으로 대체할 수 있습니다. 아래 두 예시는 정확히 동일합니다.



Javascript 코드

```
x == 0 ? 'Game Over!' : 'Keep Playing'
```

Python 코드

```
'Game Over' if x == 0 else 'Keep Playing'
```

조건문

조건문은 컴퓨터 프로그래밍에서 핵심적인 역할을 합니다. 조건문을 사용하면 다음과 같은 문장을 표현할 수 있습니다:

- 왼쪽에 길이 있다면, 왼쪽으로 돈다.
- 점수가 100 이면, “잘 했어요!”를 출력한다.

블록

만약

가장 간단한 조건문은 **만약** 블록입니다. 아래와 같이 나타낼 수 있습니다:

이 블록이 실행되면, 변수 **x** 의 값을 100 과 비교합니다.

만약 값이 100 보다 크면, “큰 수입니다.”가 출력됩니다. 그렇지 않으면 아무 일이 일어나지 않습니다.

Javascript 코드

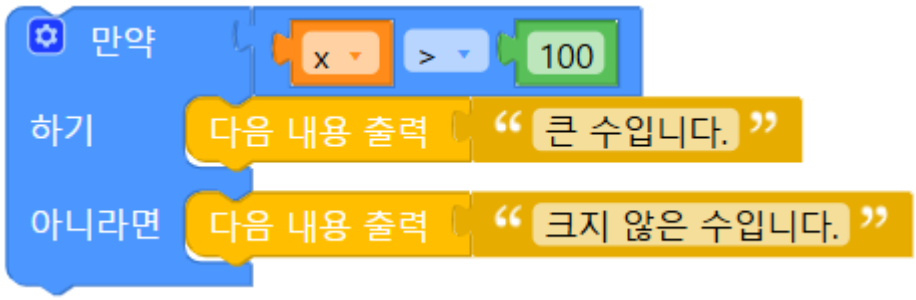
```
if(x > 100) {
    window.alert(" 큰 수입니다.");
}
```

Python 코드

```
if x > 100:
    print(" 큰 수입니다.")
```

만약-아니라면

조건이 참이 아닐 경우 무엇을 해야 할지 지정할 수도 있습니다. 아래 예시와 같습니다:



이전 블록처럼, **x** 가 100 보다 크면“큰 수입니다.”가 출력됩니다. 그렇지 않으면, “크지 않은 수입니다.”가 출력됩니다.

만약 블록에는 오직 하나의 **아니라면** 섹션이 있을 수 있으며, 그 이상은 있을 수 없습니다.

Javascript 코드

```javascript
if(x > 100) {
    window.alert('큰 수입니다.');
} else {
    window.alert('크지 않은 수입니다.');
}
```

Python 코드

```python
if x > 100:
    print('큰 수입니다.')
else:
    print('크지 않은 수입니다.')
```

만약-다른 경우

하나의 **만약** 블록에서 여러 조건을 테스트할 수도 있습니다. **다른 경우** 섹션을 추가하여 가능해집니다:

이 블록은 먼저 **x** 가 100 보다 큰지 확인하고, 그렇다면“큰 수입니다.”를 출력합니다.

그렇지 않으면, **x** 가 42 와 같은지 확인하고, 만약 그렇다면“행운의 수입니다.”를 출력합니다.

그렇지 않으면 아무 일도 일어나지 않습니다.

만약 블록은 다양한 개수의 **다른 경우** 블록을 가질 수 있습니다.

조건은 위에서 아래로 차례대로 평가되며, 하나가 만족되거나 조건이 더 이상 남아있지 않으면 끝납니다.

Javascript 코드

```javascript
if(x > 100) {
    window.alert('큰 수입니다.');
} else if(x === 42) {
    window.alert('행운의 수입니다.');
}
```

Python 코드

```python
if x > 100:
    print('큰 수입니다.')
elif x == 42:
    print('행운의 수입니다.')
```

만약-다른 경우-아니라면

아래와 같이 **만약** 블록은 **다른 경우**와 **아니라면** 섹션을 모두 가질 수 있습니다:

아니라면 섹션은 앞의 조건들이 모두 참이 아니더라도 반드시 어떤 작업을 수행하게 만듭니다.

아니라면 섹션은 어떤 수의 **다른 경우** 섹션 뒤에 올 수 있으며, 그 수는 0 이상입니다.

Javascript 코드

```javascript
if(x > 100) {
    window.alert('큰 수입니다.');
```

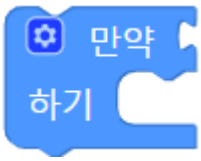
```
} else if(x === 42) {
    window.alert('행운의 수입니다.');
} else {
    window.alert('크지 않은 수입니다.');
}
```

Python 코드

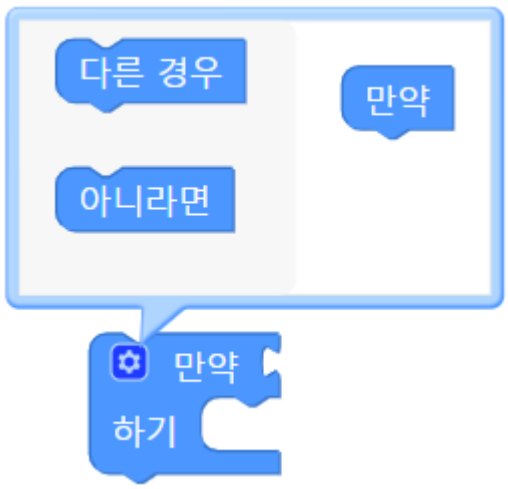
```
if x > 100:
    print('큰 수입니다.')
elif x == 42:
    print('행운의 수입니다.')
else:
    print('크지 않은 수입니다.')
```

블록 수정 방법

단순한 **만약** 블록만 도구 상자에 나타납니다:

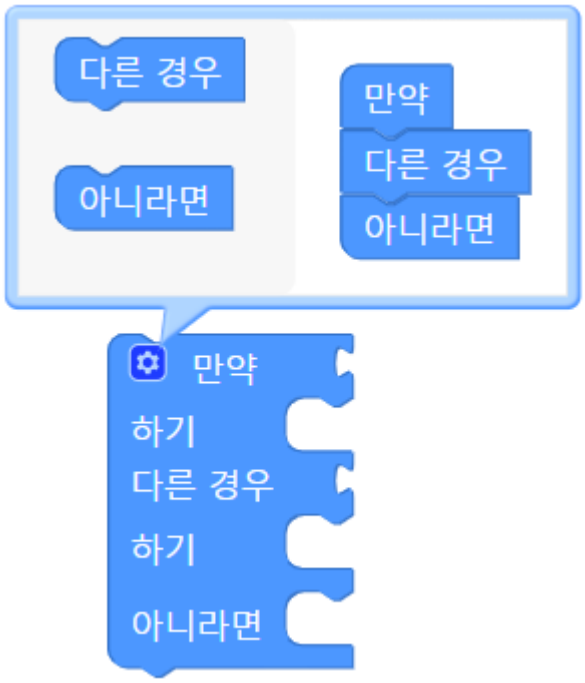


다른 경우와 **아니라면** 절을 추가하려면, 기어 아이콘을 클릭하여 새 창을 엽니다:



다른 경우와 **아니라면** 절을 **만약** 블록 아래로 드래그하고, 그것들을 재정렬하거나 제거할 수 있습니다. 완료되면 기어 아

이콘을 클릭하여 창을 닫습니다:



다른 경우 서브 블록은 원하는 개수만큼 추가할 수 있지만, 아니라면 블록은 최대 한 개만 추가할 수 있다는 점에 유의하십시오.

반복

제어문에는 두 가지 유형이 있습니다:
조건문과 **반복문** (변수들의 값에 따라 본문을 몇 번 실행할 지 제어하는 것들)

반복 생성 블록

반복

가장 간단한 **반복** 블록은 본문의 코드를 지정된 횟수만큼 실행합니다.
예를 들어, 아래의 블록은“Hello!”를 열 번 출력합니다.

Javascript 코드

```javascript
for (var count = 0; count < 10; count++) { // 10 회 반복
    window.alert('Hello');
}
```

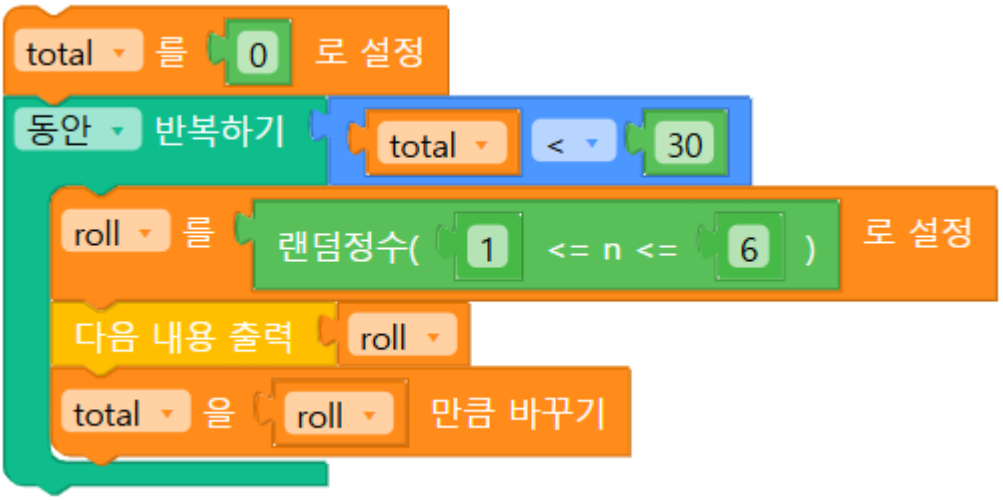
Python 코드

```python
for count in range(10): # 10 회 반복
    print('Hello')
```

동안 반복하기

플레이어가 주사위를 굴리고 총합이 30 보다 작을 때까지 값을 더하는 게임을 상상해 보세요. 아래 블록들은 그 게임을 구현한 것입니다:

1. **total** 이라는 변수가 0 으로 초기화됩니다.
2. 반복문은 **total** 이 30 보다 작은지 확인하는 것으로 시작합니다. 그렇다면 본문의 블록들이 실행됩니다.
3. 1 에서 6 까지의 랜덤 숫자가 생성되어 (주사위 굴리기) **roll** 이라는 변수에 저장됩니다.
4. 나온 숫자가 출력됩니다.
5. **total** 변수는 **roll** 만큼 증가합니다.
6. 반복문이 끝나면 제어가 2 단계로 돌아갑니다.



반복문이 완료되면 그 이후의 블록들이 실행됩니다 (생략된 부분).

이 예시에서는 1 에서 6 까지의 랜덤 숫자가 몇 번 출력된 후 반복문은 종료되고, **total** 변수에는 그 합이 저장됩니다.

이 합은 최소한 30 이 됩니다.

Javascript 코드

```javascript
var roll, total;

function mathRandomInt(a, b) { // 무작위 수 함수
    if (a > b) {
        var c = a;
        a = b;
        b = c;
    }
    return Math.floor(Math.random() * (b - a + 1) + a);
}

total = 0;
while(total < 30) { // 30 이하인 동안 반복하기
    roll = mathRandomInt(1, 6);
    window.alert(roll);
    total = (typeof total === 'number' ? total : 0) + roll;
    await __wait(10);
}
```

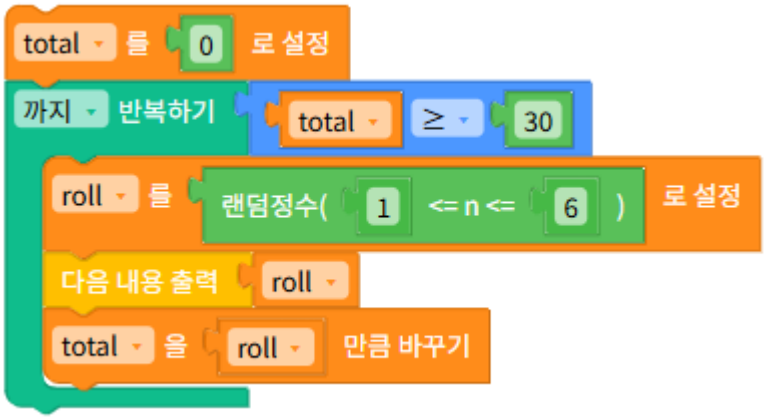
Python 코드

```python
import asyncio
import random
from numbers import Number

roll = None
total = None

total = 0
while total < 30: # 30 이하인 동안 반복하기
    roll = random.randint(1, 6)
    print(roll)
```

```
    total = (total if isinstance(total, Number) else 0) + roll
    await asyncio.sleep(0.01)
```

까지 반복하기

까지 반복하기 반복문은 조건이 거짓인 동안 본문을 반복하고, 조건이 참이 되는 순간 반복문을 탈출합니다.
아래 블록들은 이전 예시와 동일한 결과를 도출합니다.
반복문은 **total** 이 30 이상일 때까지 반복됩니다.



Javascript 코드

```
var total, roll;

function mathRandomInt(a, b) {
    if (a > b) { // 무작위 수 함수
        var c = a;
        a = b;
        b = c;
    }
    return Math.floor(Math.random() * (b - a + 1) + a);
}

total = 0;
while(!(total >= 30)) { // 30 이상이 될 때까지 반복하기
    roll = mathRandomInt(1, 6);
    window.alert(roll);
```

```
    total = (typeof total === 'number' ? total : 0) + roll;
    await __wait(10);
}
```

Python 코드

```
import asyncio
import random
from numbers import Number

total = None
roll = None

total = 0
while not (total >= 30): # 30 이상이 될 때까지 반복하기
    roll = random.randint(1, 6)
    print(roll)
    total = (total if isinstance(total, Number) else 0) + roll
    await asyncio.sleep(0.01)
```

으로 계산

으로 계산 블록 (대부분 **for loop** 라고 부름) 은 변수를 첫 번째 값에서 세 번째 값까지 증가량 (두 번째 값) 만큼 증가시키면서 본문을 각 값에 대해 한 번씩 실행합니다.

예를 들어, 아래 프로그램은 1, 3, 5 를 출력합니다.

Javascript 코드

```
var index;
```

```javascript
for (index = 1; index <= 5; index += 2) {
    window.alert(index);
}
```

Python 코드

```python
index = None

for index in range(1, 6, 2):
    print(index)
```

다음 두 반복문은 첫 번째 입력이 두 번째 입력보다 클 수도 있습니다. 세 번째 값 (증가량) 이 양수인지 음수인지에 관계없이 동일한 동작을 합니다.

Javascript 코드

```javascript
var index;

for (index = 5; index >= 1; index -= 2) {
    window.alert(index);
}
```

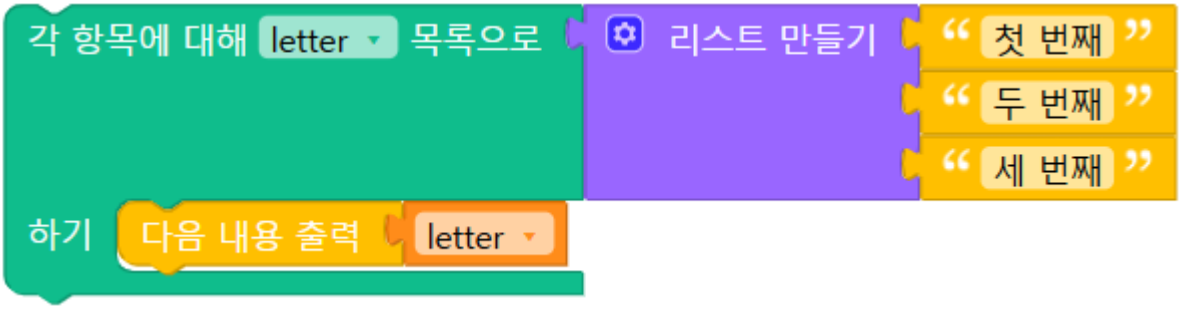
Python 코드

```
index = None

for index in range(5, 0, -2):
    print(index)
```

각 항목에 대해

각 항목에 대해 블록은 숫자 순서 대신 목록의 값을 차례대로 사용하는 점에서 유사합니다.

아래 프로그램은“첫 번째”, “두 번째”, “세 번째”목록의 각 항목을 출력합니다.



Javascript 코드

```
var letter;

var letter_list = ['첫 번째', '두 번째', '세 번째'];
for (var letter_index in letter_list) { // 각 항목에 대해
    letter = letter_list[letter_index];
    window.alert(letter);
}
```

Python 코드

```
letter = None

for letter in ['첫 번째', '두 번째', '세 번째']: # 각 항목에 대해
    print(letter)
```

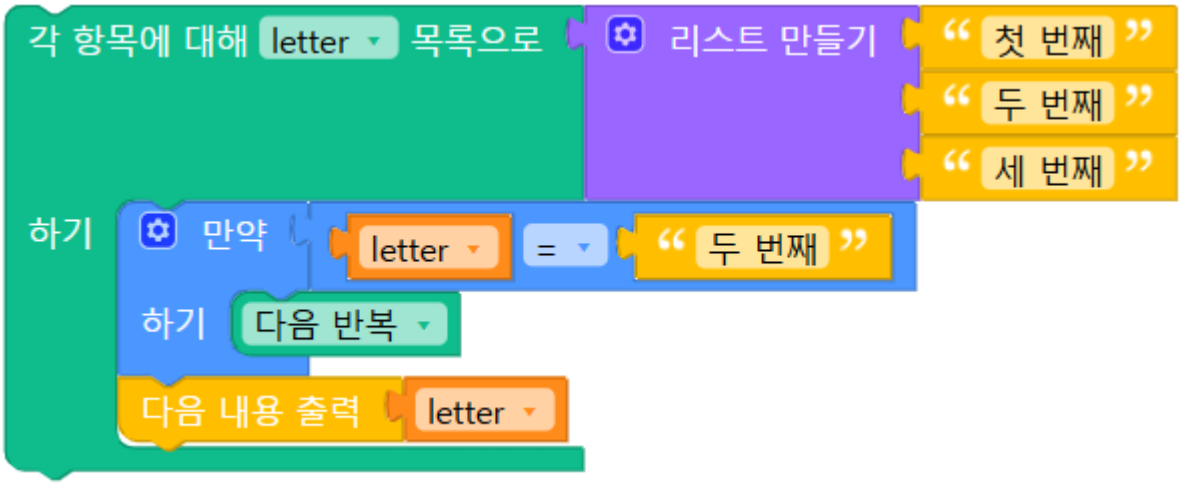
반복문 종료 블록

대부분의 반복문은 **종료 조건이 충족될 때까지** (반복 블록의 경우) 또는 **반복문 변수에 의해 값이 모두 처리될 때까지** 실행됩니다. (**으로 계산**과 **각 항목에 대해** 블록의 경우)
드물게 사용되지만 유용할 수 있는 두 가지 블록은 반복문 동작을 제어하는 추가적인 방법을 제공합니다.
아래 예시는 **각 항목에 대해** 반복문에 해당하지만, 모든 유형의 반복문에 사용할 수 있습니다.

다음 반복

다음 반복 (대부분의 프로그래밍 언어에서 **continue**) 은 본문의 남은 코드를 건너뛰고 다음 반복 (패스) 을 시작하게 만듭니다.

다음 프로그램은 반복문은 첫 번째 반복에서“첫 번째”를 출력합니다.
두 번째 반복에서는 **다음 반복** 블록이 실행되어“두 번째”출력이 건너뜁니다.
마지막 반복에서“세 번째”가 출력됩니다.



Javascript 코드

```
var letter;

var letter_list = ['첫 번째', '두 번째', '세 번째'];
for (var letter_index in letter_list) {
    letter = letter_list[letter_index];
    if (letter == '두 번째') {
        continue; // 다음 반복
    }
    window.alert(letter);
```

```
}
```

Python 코드

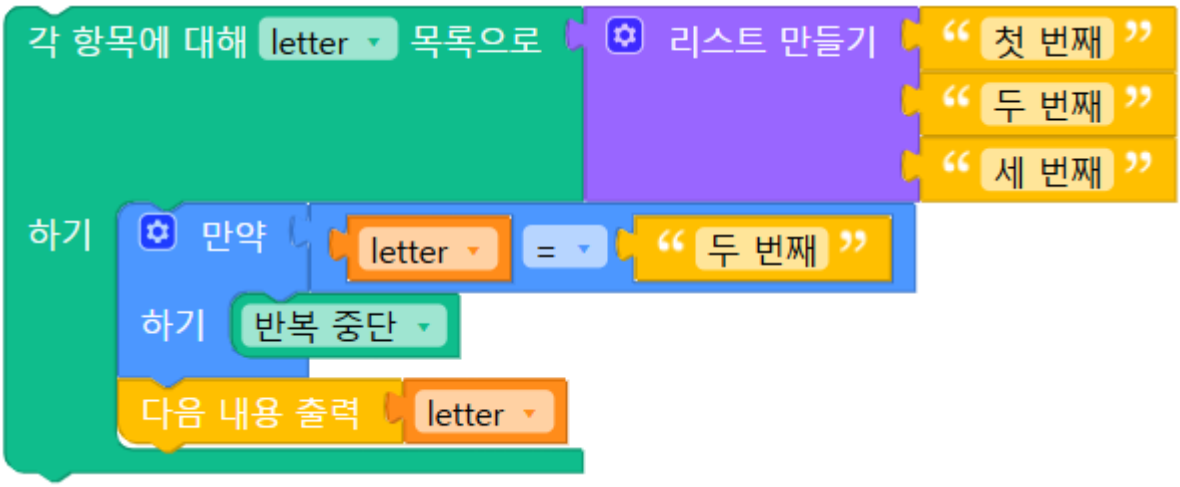
```
letter = None

for letter in ['첫 번째', '두 번째', '세 번째']:
    if letter == '두 번째':
        continue # 다음 반복
    print(letter)
```

반복 중단

반복 중단 블록은 **반복문에서 일찍 빠져나올 수** 있게 해줍니다.

아래 프로그램은 첫 번째 반복에서“첫 번째”를 출력하고, 두 번째 반복에서 반복문은 변수 값이“두 번째”일 때 **반복 중단** 블록이 실행되어 반복문에서 빠져나옵니다.

리스트의“세 번째”항목은 출력되지 않습니다.



Javascript 코드

```
var letter;

var letter_list = ['첫 번째', '두 번째', '세 번째'];
for (var letter_index in letter_list) {
    letter = letter_list[letter_index];
```

```
    if (letter == '두 번째') {
        break; // 반복 중단
    }
    window.alert(letter);
}
```

Python 코드

```
letter = None

for letter in ['첫 번째', '두 번째', '세 번째']:
    if letter == '두 번째':
        break # 반복 중단
    print(letter)
```

연산

이 문서는 다양한 연산 블록의 기능과 사용법을 설명합니다.
숫자 연산, 리스트 처리, 확률 및 각도 연산 등 다양한 수학적 연산을 수행하는 블록을 소개합니다.

블록

숫자 값

입력된 **숫자 값**을 그대로 반환하는 블록입니다. 이 블록을 사용하면 특정 숫자를 변수에 저장하거나 다른 연산에 활용할 수 있습니다.

Javascript 코드

```
10; // 숫자 값 10
```

Python 코드

```
10 # 숫자 값 10
```

배열 생성 및 연산

배열을 생성하는 블록입니다.

[] 안에 입력한 값들을 요소로 가지는 배열을 반환합니다.

[] 안에 원하는 값을 입력하여 리스트를 만들 수 있으며, 문자열은”“로 감싸야 합니다.

위의 블록은 [“하나”, “둘”, 3] 을 생성하며, 하나, 둘, 3 을 순서대로 출력합니다.

Javascript 코드

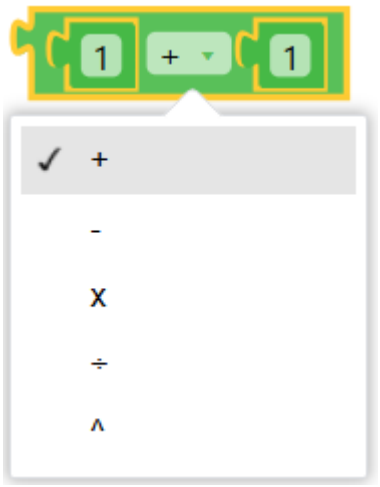
```
var item;

var item_list = [" 하나", " 둘", 3 ]; // 배열 생성
for (var item_index in item_list) {
    item = item_list[item_index];
    window.alert(item);
}
```

Python 코드

```
item = None

for item in [" 하나", " 둘", 3 ]: # 배열 생성
    print(item)
```

기본 산술 연산

두 개의 숫자 값을 사용하여 **산술 연산** (덧셈, 뺄셈, 곱셈, 나눗셈, 거듭제곱) 을 수행하는 블록입니다.



위의 블록은 1 + 1 을 계산하여 2 를 반환합니다.

Javascript 코드

```
1 + 1; // 1 + 1 연산
1 - 1; // 1 - 1 연산
1 * 1; // 1 * 1 연산
1 / 1; // 1 / 1 연산
Math.pow(1, 1); // 1^1 연산
```

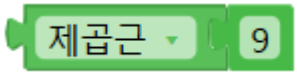
Python 코드

```
1 + 1 # 1 + 1 연산
1 - 1 # 1 - 1 연산
1 * 1 # 1 * 1 연산
1 / 1 # 1 / 1 연산
1 ** 1 # 1^1 연산
```

고급 연산

제곱근

입력된 숫자의 **제곱근**을 반환합니다.



위 블록은 9 의 제곱근을 구해 3 을 반환합니다.

Javascript 코드

```
Math.sqrt(9); // 9 의 제곱근
```

Python 코드

```
import math

math.sqrt(9) # 9 의 제곱근
```

절댓값

입력된 숫자의 **절댓값**을 반환합니다.

다음 블록의 값은 10 입니다.

Javascript 코드

```
Math.abs(-10); // -10 의 절댓값
```

Python 코드

```
import math

math.fabs(-10) # -10 의 절댓값
```

부호

입력된 숫자의 **부호를 반대로 변경**합니다.

다음 블록의 값은 -10 입니다.

Javascript 코드

```
-10; // 10 의 반대 부호 값
```

Python 코드

```
import math

-10 # 10 의 반대 부호 값
```

지수, 로그 함수

지수와 로그 값을 계산하는 블록입니다. 특정 밑수의 거듭제곱이나 로그 값을 구할 때 사용됩니다.

Javascript 코드

```
Math.log(10); // 자연 로그 10 의 값
Math.log(10)/ Math.log(10); // 밑이 10, 진수가 10 인 로그 값
Math.exp(2); // e 의 제곱 값
Math.pow(10,2); // 10 의 제곱 값
```

Python 코드

```
import math

math.log(10) # 자연 로그 10 의 값
math.log10(10) # 밑이 10, 진수가 10 인 로그 값
```

```
math.exp(2) # e 의 제곱 값
math.pow(10,2) # 10 의 제곱 값
```

나머지

두 숫자의 나눗셈에서 **나머지**를 구하는 블록입니다.

Javascript 코드

```
64 % 10; // 64 ÷ 10 의 나머지 값
```

Python 코드

```
64 % 10 # 64 ÷ 10 의 나머지 값
```

삼각 함수

사인, 코사인, 탄젠트 등 **삼각 함수** 값을 계산하는 블록입니다.

Javascript 코드

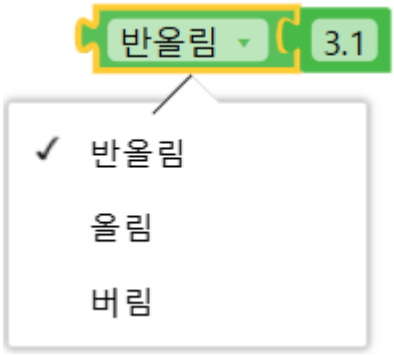
```
Math.sin(45 / 180 * Math.PI); // sin 45° 값
Math.cos(45 / 180 * Math.PI); // cos 45° 값
Math.tan(45 / 180 * Math.PI); // tan 45° 값
Math.asin(45) / Math.PI * 180; // asin 45° 값
Math.acos(45) / Math.PI * 180; // acos 45° 값
Math.atan(45) / Math.PI * 180; // atan 45° 값
```

Python 코드

```
import math
math.sin(45 / 180.0 * math.pi) # sin 45° 값
math.cos(45 / 180.0 * math.pi) # cos 45° 값
math.tan(45 / 180.0 * math.pi) # tan 45° 값
math.asin(45) / math.pi * 180 # asin 45° 값
math.acos(45) / math.pi * 180 # acos 45° 값
math.atan(45) / math.pi * 180 # atan 45° 값
```

올림, 버림, 반올림

입력된 숫자를 올림, 버림, 반올림을 수행하여 값을 반환하는 블록입니다.



Javascript 코드

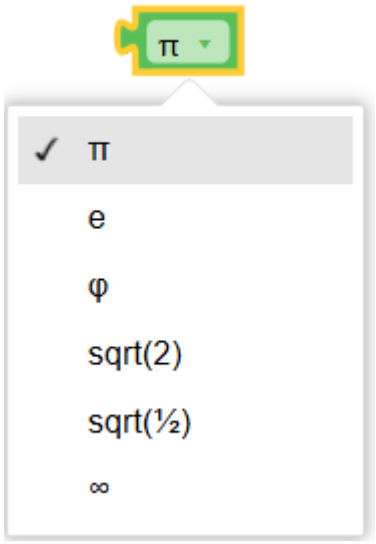
```
Math.round(3.1); // 3.1 반올림 값 3
Math.ceil(3.1); // 3.1 올림 값 4
Math.floor(3.1); // 3.1 버림 값 3
```

Python 코드

```
import math

round(3.1) # 3.1 반올림 값 3
math.ceil(3.1) # 3.1 올림 값 4
math.floor(3.1) # 3.1 버림 값 3
```

특수 숫자 값

연산에 필요한 특수한 **수학 상수** (π, e 등) 와 기호를 제공하는 블록입니다.



Javascript 코드

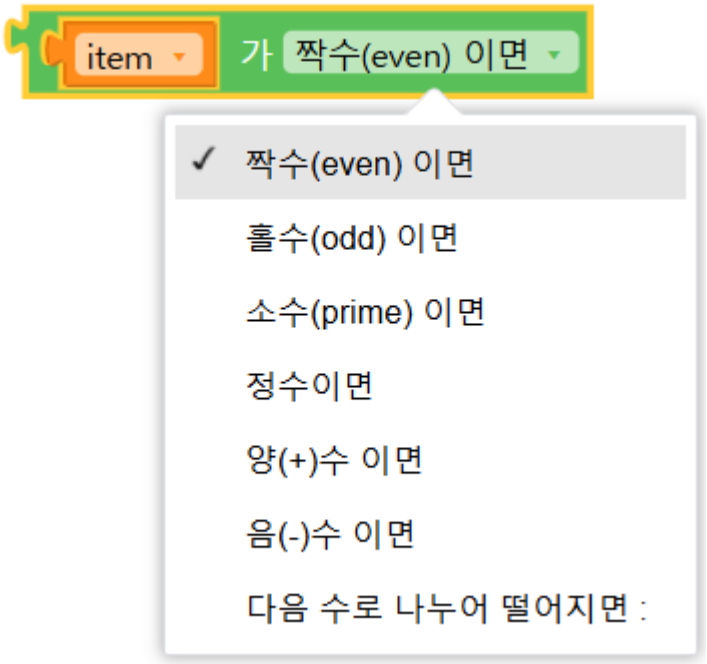
```
Math.PI; // π(원주율) 값
Math.E; // e(자연상수) 값
(1 + Math.sqrt(5)) / 2; // 황금비 값
Math.SQRT2; // 2 의 제곱근 값
Math.SQRT1_2; // 1/2 의 제곱근 값
Infinity; // 무한대 값
```

Python 코드

```
import math


math.pi # π(원주율) 값
math.e # e(자연상수) 값
(1 + math.sqrt(5)) / 2 # 황금비 값
math.sqrt(2) # 2 의 제곱근 값
math.sqrt(1.0 / 2) # 1/2 의 제곱근 값
float('inf') # 무한대 값
```

숫자 조건

입력된 숫자 값이 특정 조건을 만족하는지 검사하는 블록입니다.
숫자가 짝수인지, 홀수인지, 소수인지 등을 판별할 수 있습니다.
조건의 성립 여부를 검사하고 성립하면 **참**을 성립하지 않으면 **거짓**을 나타냅니다.



Javascript 코드

```
var item;


function mathIsPrime(n) { // 소수 판단 함수
```

```
    if (n == 2 || n == 3) {
        return true;
    }
    if (isNaN(n) || n <= 1 || n % 1 !== 0 || n % 2 === 0 || n % 3 === 0) {
        return false;
    }
    for (var x = 6; x <= Math.sqrt(n) + 1; x += 6) {
        if (n % (x - 1) === 0 || n % (x + 1) === 0) {
            return false;
        }
    }
    return true;
}

item % 2 === 0; // item 이 짝수인지 조건 성립 여부
item % 2 === 1; // item 이 홀수인지 조건 성립 여부
mathIsPrime(item); // item 이 소수인지 조건 성립 여부
item % 1 === 0; // item 이 정수인지 조건 성립 여부
item > 0; // item 이 양수인지 조건 성립 여부
item < 0; // item 이 음수인지 조건 성립 여부
item % 0 === 0 // item 이 0 으로 나누어 떨어지는지 조건 성립 여부
```

Python 코드

```
import math
from numbers import Number

item = None

def math_isPrime(n): # 소수 판단 함수
    if not isinstance(n, Number):
        try:
            n = float(n)
        except:
```

```
            return False
    if n == 2 or n == 3:
        return True
    if n <= 1 or n % 1 != 0 or n % 2 == 0 or n % 3 == 0:
        return False
    for x in range(6, int(math.sqrt(n)) + 2, 6):
        if n % (x - 1) == 0 or n % (x + 1) == 0:
            return False
    return True

item % 2 == 0 # item 이 짝수인지 조건 성립 여부
item % 2 == 1 # item 이 홀수인지 조건 성립 여부
math_isPrime(item) # item 이 소수인지 조건 성립 여부
item % 1 == 0 # item 이 정수인지 조건 성립 여부
item > 0 # item 이 양수인지 조건 성립 여부
item < 0 # item 이 음수인지 조건 성립 여부
item % 0 == 0  # item 이 0 으로 나누어 떨어지는지 조건 성립 여부
```

아래 예제는 item 이 짝수이므로, “짝수입니다.”라는 메세지가 출력됩니다.

Javascript 코드

```
var item;

item = 2;
if (item % 2 === 0) {
```

```
    window.alert('짝수입니다.');
} else {
    window.alert('홀수입니다.');
}
```

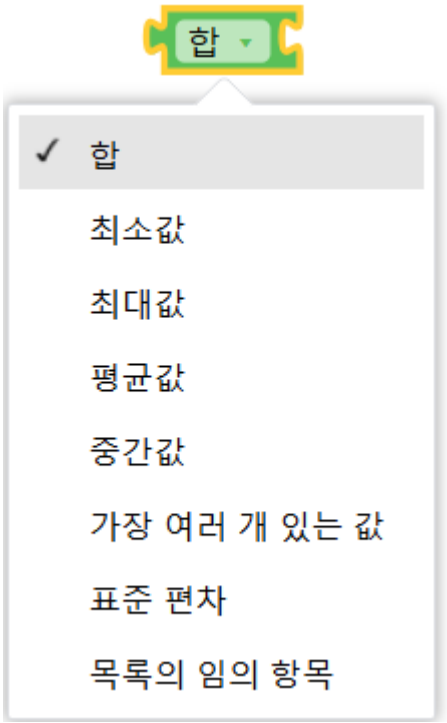
Python 코드

```
item = None

item = 2
if item % 2 == 0:
    print('짝수입니다.')
else:
    print('홀수입니다.')
```

리스트 연산

리스트를 대상으로 다양한 연산을 수행하는 블록입니다.

합계, 최솟값, 최댓값, 평균값, 중간값, 가장 여러 개 있는 값, 표준 편차, 임의의 항목을 구할 수 있습니다.



Javascript 코드

```
function mathMean(myList) { // 평균 계산 함수
    return myList.reduce(function(x, y) {return x + y;}, 0) / myList.length;
}

function mathMedian(myList) { // 중간값 계산 함수
    var localList = myList.filter(function (x) {return typeof x === 'number';});
    if (!localList.length) return null;
    localList.sort(function(a, b) {return b - a;});
    if (localList.length % 2 === 0) {
        return (localList[localList.length / 2 - 1] + localList[localList.length / 2]) / 2;
    } else {
        return localList[(localList.length - 1) / 2];
    }
}

function mathModes(values) { // 최대 빈도수 값 계산 함수
    var modes = [];
    var counts = [];
    var maxCount = 0;
    for (var i = 0; i < values.length; i++) {
        var value = values[i];
        var found = false;
        var thisCount;
        for (var j = 0; j < counts.length; j++) {
            if (counts[j][0] === value) {
                thisCount = ++counts[j][1];
                found = true;
                break;
            }
        }
        if (!found) {
            counts.push([value, 1]);
            thisCount = 1;
```

```
        }
        maxCount = Math.max(thisCount, maxCount);
    }
    for (var j = 0; j < counts.length; j++) {
        if (counts[j][1] === maxCount) {
            modes.push(counts[j][0]);
        }
    }
    return modes;
}

function mathStandardDeviation(numbers) { // 표준편차 계산 함수
    var n = numbers.length;
    if (!n) return null;
    var mean = numbers.reduce(function(x, y) {return x + y;}) / n;
    var variance = 0;
    for (var j = 0; j < n; j++) {
        variance += Math.pow(numbers[j] - mean, 2);
    }
    variance /= n;
    return Math.sqrt(variance);
}

function mathRandomList(list) { // 무작위 수 계산 함수
    var x = Math.floor(Math.random() * list.length);
    return list[x];
}


[].reduce(function(x, y) {return x + y;}, 0); // 리스트 합
Math.min.apply(null, []); // 리스트 최소 값
Math.max.apply(null, []); // 리스트 최대 값
mathMean([]); // 리스트 평균 값
mathMedian([]); // 리스트 중간 값
```

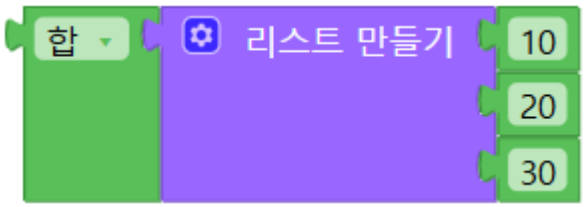
```
mathModes([]); // 리스트 최대 빈도수 값
mathStandardDeviation([]); // 리스트 표준편차 값
mathRandomList([]); // 리스트 무작위 수 값
```

Python 코드

```
from numbers import Number
import math
import random

item = None

def math_mean(myList): # 평균 계산 함수
    localList = [e for e in myList if isinstance(e, Number)]
    if not localList: return
    return float(sum(localList)) / len(localList)

def math_median(myList): # 중간값 계산 함수
    localList = sorted([e for e in myList if isinstance(e, Number)])
    if not localList: return
    if len(localList) % 2 == 0:
        return (localList[len(localList) // 2 - 1] + localList[len(localList) // 2]) / 2.0
    else:
        return localList[(len(localList) - 1) // 2]

def math_modes(some_list): # 최대 빈도수 값 계산 함수
    modes = []
    counts = []
    maxCount = 1
    for item in some_list:
        found = False
        for count in counts:
            if count[0] == item:
                count[1] += 1
```

```
                maxCount = max(maxCount, count[1])
                found = True
        if not found:
            counts.append([item, 1])
    for counted_item, item_count in counts:
        if item_count == maxCount:
            modes.append(counted_item)
    return modes

def math_standard_deviation(numbers): # 표준편차 계산 함수
    n = len(numbers)
    if n == 0: return
    mean = float(sum(numbers)) / n
    variance = sum((x - mean) ** 2 for x in numbers) / n
    return math.sqrt(variance)

sum([]) # 리스트 합
min([]) # 리스트 최소 값
max([]) # 리스트 최대 값
math_mean([]) # 리스트 평균 값
math_median([]) # 리스트 중간 값
math_modes([]) # 리스트 최대 빈도수 값
math_standard_deviation([]) # 리스트 표준편차 값
random.choice([]) # 리스트 무작위 수 값
```

아래 블록은 [10, 20, 30] 리스트의 합을 계산하는 블록으로, 10 + 20 + 30 = 60 이 됩니다.



Javascript 코드

```
[10, 20, 30].reduce(function(x, y) {return x + y;}, 0); // [10, 20, 30] 리스트 합의 결과값 60
```

Python 코드

```
sum([10, 20, 30]) # [10, 20, 30] 리스트 합의 결과값 60
```

값의 범위 설정

입력된 값이 특정 범위를 벗어나지 않도록 제한하는 블록입니다. - 최솟값보다 작으면 최솟값으로 조정됩니다. - 최댓값보다 크면 최댓값으로 조정됩니다.

item 의 값이 1 보다 작으면 1 을, 100 보다 크면 100 을 반환합니다.
1 과 100 사이의 값은 동일한 값을 반환합니다.

Javascript 코드

```
var item;

Math.min(Math.max(item, 1), 100); // item 을 1 ~ 100 사이로 조정
```

Python 코드

```
item = None

min(max(item, 1), 100) # item 을 1 ~ 100 사이로 조정
```

랜덤 정수

지정한 범위 내에서 **랜덤한 정수**를 생성하는 블록입니다.

1 이상 100 이하의 정수를 생성합니다.

Javascript 코드

```
function mathRandomInt(a, b) { // 랜덤 정수 생성 함수
    if (a > b) {
        var c = a;
        a = b;
        b = c;
    }
    return Math.floor(Math.random() * (b - a + 1) + a);
}

mathRandomInt(1, 100); // 1 ~ 100 사이 랜덤 정수 생성
```

Python 코드

```
import random

random.randint(1, 100) # 1 ~ 100 사이 랜덤 정수 생성
```

임의 분수

0 과 1 사이에서 **무작위의 분수 값**을 생성하는 블록입니다.

Javascript 코드

```
Math.random(); // 무작위의 분수 값
```

Python 코드

```
import random
```

```
random.random() # 무작위의 분수 값
```

각도

주어진 (x, y) 좌표가 원점 (0,0) 과 이루는 각도를 계산하는 블록입니다. 좌표 위치를 기반으로 방향을 판별하는 데 사용할 수 있습니다.

Javascript 코드

```
Math.atan2(1, 1) / Math.PI * 180; // 각도 계산 값
```

Python 코드

```
import math

math.atan2(1, 1) / math.pi * 180 # 각도 계산 값
```

문자열

문자열의 예시는 다음과 같습니다: - “thing #1”- “March 12, 2010”- “”(빈 문자열)

문자열에는 대문자 또는 소문자로 된 문자, 숫자, 구두점, 기타 기호 및 단어 사이의 공백이 포함될 수 있습니다.

블록

문자열 만들기

다음 블록은“hello”라는 문자열를 생성하고 이를 greeting 이라는 변수에 저장합니다.

Javascript 코드

```
var greeting;

greeting = 'hello';
```

Python 코드

```
greeting = None

greeting = 'hello'
```

문자열 만들기 블록은 greeting 변수의 값과 새로운 문자열“world”를 결합 (연결) 하여“helloworld”라는 문자열을 생성합니다.
원래 문자열 중 어느 것도 공백을 포함하지 않았기 때문에 공백이 없습니다.

Javascript 코드

```
var greeting;

greeting = 'hello';
String(greeting) + 'world'; // "helloworld"
```

Python 코드

```
greeting = None

greeting = 'hello'
str(greeting) + 'world' # "helloworld"
```

입력 문자열의 개수를 늘리려면 톱니바퀴 아이콘을 클릭하면 다음과 같이 보기가 변경됩니다:

추가 입력을 추가하려면 왼쪽의 회색 도구 상자에서 **항목** 블록을 끌어와 **가입** 블록에 삽입하면 됩니다.

문자열 수정

…**내용 덧붙이기** 블록은 지정된 변수에 주어진 문자열를 추가합니다.

아래 greeting 변수의 값이“hello”에서“hello, there!”로 변경됩니다.

Javascript 코드

```
var greeting;

greeting = 'hello';
greeting += ', there!'; // greeting = "hello, there!"
```

Python 코드

```
greeting = None

greeting = 'hello'
greeting = str(greeting) + ', there!' # greeting = "hello, there!"
```

문자열 길이

길이 블록은 각 문자열에서 문자, 숫자 등을 세어 총 길이를 반환합니다.
아래“We're #1!”의 길이는 9 이며, 빈 문자열의 길이는 0 입니다.

Javascript 코드

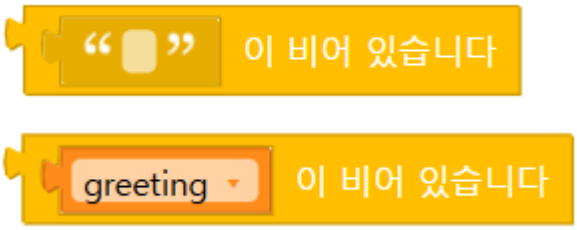
```
'We\'re #1!'.length; // 'We're #1!'의 문자열 길이 값 9 반환
''.length // ''의 문자열 길이 값 0 반환
```

Python 코드

```
len("We're #1!") # "We're #1!" 의 문자열 길이 값 9 반환
len('') # ''의 문자열 길이 값 0 반환
```

빈 문자열 확인

비어 있습니다 블록은 주어진 문자열가 비었는지 (길이가 0 인지) 확인합니다.
첫 번째 경우에는 **참**이 반환되고, 두 번째 경우에는 **거짓**이 반환됩니다.



Javascript 코드

```
var greeting;

greeting = 'hello';
!''.length; // 문자열이 비어 있으므로 참 반환
!greeting.length; // 문자열이 비어있지 않으므로 거짓 반환
```

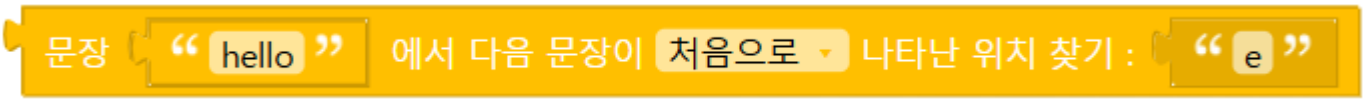
Python 코드

```
greeting = None

greeting = 'hello'
not len('') # 문자열이 비어 있으므로 참 반환
not len(greeting) # 문자열이 비어있지 않으므로 거짓 반환
```

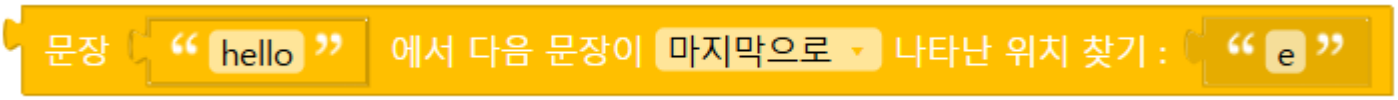
문자열 찾기

이 블록들은 특정 문자열가 다른 문자열 안에 있는지 확인하고, 존재하는 경우 위치를 반환합니다.

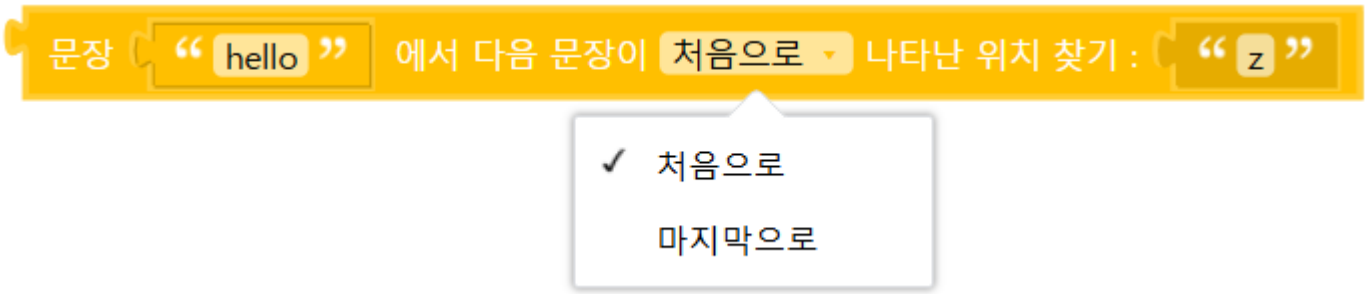
예를 들어, 아래 블록은“hello”에서“e”의 첫 번째 등장 위치를 요청하며 결과는 2 입니다.



아래 블록은“hello”에서“e”의 마지막 등장 위치를 요청하며 결과는 동일하게 2 입니다.



첫 번째 또는 **마지막** 옵션이 선택되더라도“hello”에는“z”가 포함되지 않으므로 결과는 0 이 됩니다.



Javascript 코드

```
'hello'.indexOf('e') + 1; // "hello" 문자열에서 처음으로 'e'가 나타난 위치 값 2 반환
'hello'.lastIndexOf('e') + 1; // "hello" 문자열에서 마지막으로 'e'가 나타난 위치 값 2 반환
'hello'.indexOf('z') + 1; // "hello" 문자열에서 처음으로 'z'가 나타난 위치 값 0 반환
```

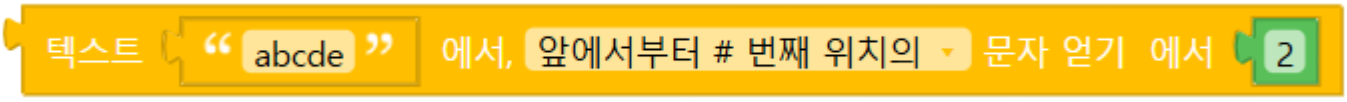
Python 코드

```
'hello'.find('e') + 1 # "hello" 문자열에서 처음으로 'e'가 나타난 위치 값 2 반환
'hello'.rfind('e') + 1 # "hello" 문자열에서 마지막으로 'e'가 나타난 위치 값 2 반환
'hello'.find('z') + 1 # "hello" 문자열에서 처음으로 'z'가 나타난 위치 값 0 반환
```

문자열 추출

단일 문자 추출

이 블록은“abcde”에서 두 번째 문자“b”를 가져옵니다:

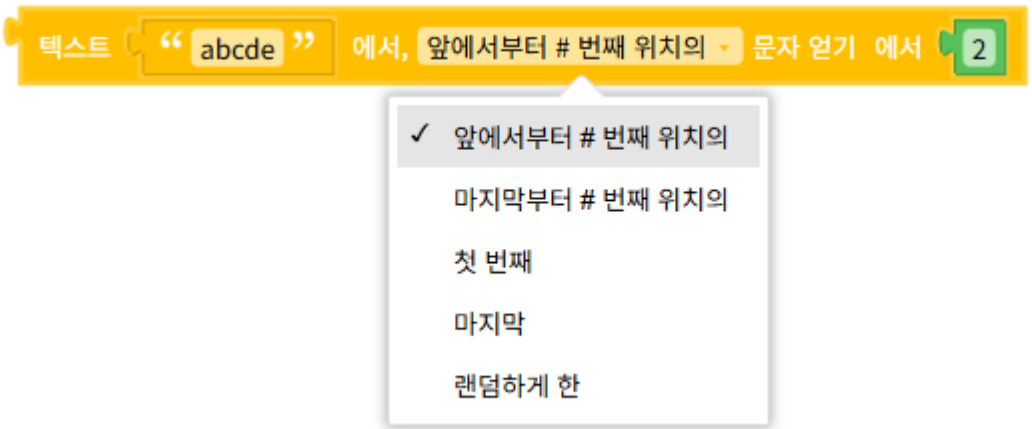


Javascript 코드

```
'abcde'.charAt(1); // "abcde" 에서 두 번째 문자 'b' 반환
```

Python 코드

```
'abcde'[1] # "abcde" 에서 두 번째 문자 'b' 반환
```



Javascript 코드

```javascript
function textRandomLetter(text) { // 무작위 문자 반환 함수
    var x = Math.floor(Math.random() * text.length);
    return text[x];
}

'abcde'.charAt(1); // 앞에서부터 두 번째 위치의 문자 'b' 얻기
'abcde'.slice(-2).charAt(0); // 마지막부터 두 번째 위치의 문자 'd' 얻기
'abcde'.charAt(0); // 첫 번째 문자 'a' 얻기
'abcde'.slice(-1); // 마지막 문자 'e' 얻기
textRandomLetter('abcde'); // 랜덤하게 한 문자 얻기
```

Python 코드

```python
def text_random_letter(text): # 무작위 문자 반환 함수
    x = int(random.random() * len(text))
    return text[x]

'abcde'[1] # 앞에서부터 두 번째 위치의 문자 'b' 얻기
'abcde'[-2] # 마지막부터 두 번째 위치의 문자 'd' 얻기
'abcde'[0] # 첫 번째 문자 'a' 얻기
'abcde'[-1] # 마지막 문자 'e' 얻기
text_random_letter('abcde') # 랜덤하게 한 문자 얻기
```

문자열 일부 추출

문자열에서…부분 문자열 가져오기 블록을 사용하면 특정 범위의 문자열를 추출할 수 있습니다.

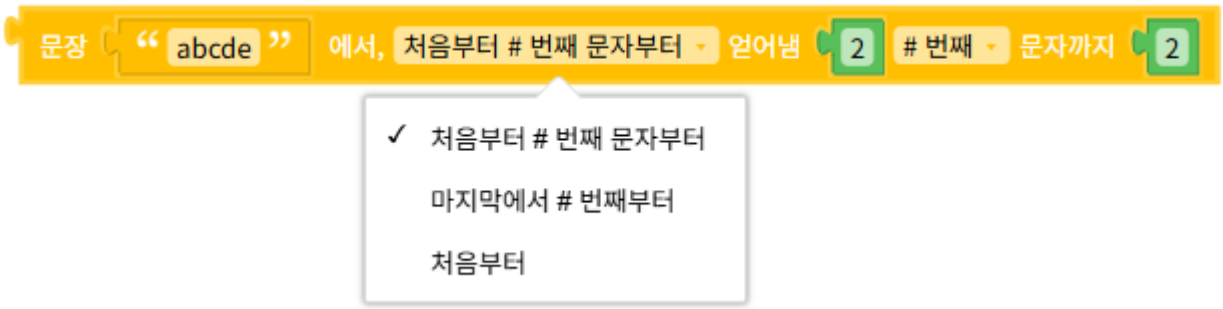
위 블록은“abc”를 추출합니다.

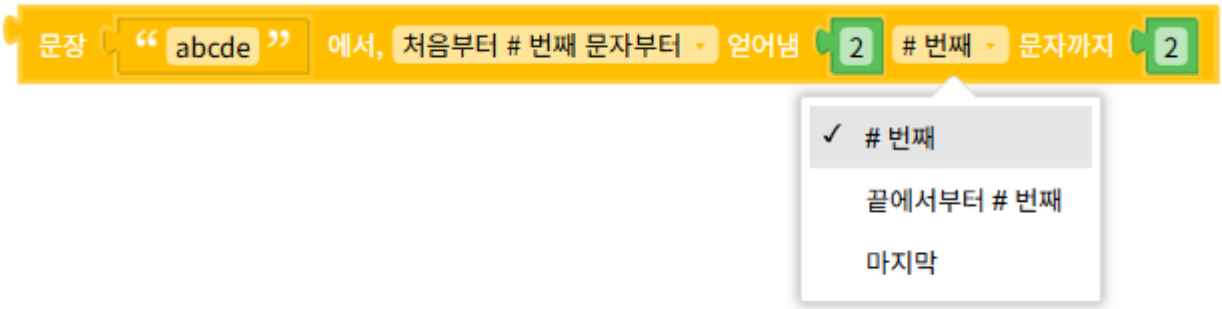
Javascript 코드

```
'abcde'.slice(0, 3); // "abcde" 에서 처음 ~ 세 번째 문자까지의 문자열 "abc" 반환
```

Python 코드

```
'abcde'[ : 3] # "abcde" 에서 처음 ~ 세 번째 문자까지의 문자열 "abc" 반환
```





Javascript 코드

```
'abcde'.slice(1, 4); // 'abcde' 처음부터 2 번째 문자부터 4 번째 문자까지의 문자열 'bcd' 추출
'abcde'.slice(1, 'abcde'.length - 1); // 'abcde' 처음부터 2 번째 문자부터 끝에서부터 2 번째 문자까지의 문자열 'bcd' 추출
'abcde'.slice(1, 'abcde'.length); // 'abcde' 처음부터 2 번째 문자부터 마지막 문자까지 문자열 'bcde' 추출
'abcde'.slice('abcde'.length - 4, 4); // 'abcde' 마지막에서 4 번째 문자부터 4 번째 문자까지의 문자열 'bcd' 추출
'abcde'.slice('abcde'.length - 4, 'abcde'.length - 1); // 'abcde' 마지막에서 4 번째 문자부터 끝에서부터 2 번째 문자까지의 문자열 'bcd' 추출
'abcde'.slice('abcde'.length - 4, 'abcde'.length); // 'abcde' 마지막에서 4 번째 문자부터 마지막 문자까지의 문자열 'bcde' 추출
'abcde'.slice(0, 4); // 'abcde' 처음부터 4 번째 문자까지의 문자열 'abcd' 추출
```

```
'abcde'.slice(0, 'abcde'.length - 1); // 'abcde' 처음부터 끝에서부터 2 번째 문자까지의 문자열 'abcd' 추출
'abcde'; // 'abcde' 처음부터 마지막 문자까지의 문자열 'abcde' 추출
```

Python 코드

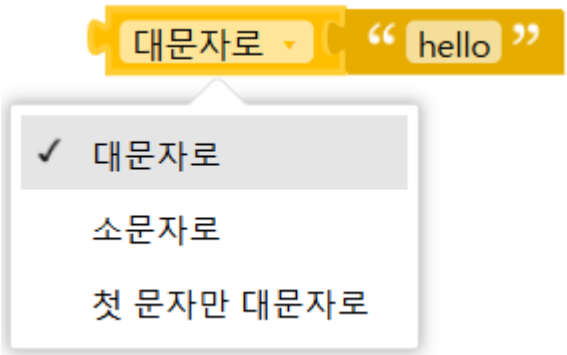
```
'abcde'[1 : 4] # 'abcde' 처음부터 2 번째 문자부터 4 번째 문자까지의 문자열 'bcd' 추출
'abcde'[1 : -1] # 'abcde' 처음부터 2 번째 문자부터 끝에서부터 2 번째 문자까지의 문자열 'bcd' 추출
'abcde'[1 : ] # 'abcde' 처음부터 2 번째 문자부터 마지막 문자까지 문자열 'bcde' 추출
'abcde'[-4 : 4] # 'abcde' 마지막에서 4 번째 문자부터 4 번째 문자까지의 문자열 'bcd' 추출
'abcde'[-4 : -1] # 'abcde' 마지막에서 4 번째 문자부터 끝에서부터 2 번째 문자까지의 문자열 'bcd' 추출
'abcde'[-4 : ] # 'abcde' 마지막에서 4 번째 문자부터 마지막 문자까지의 문자열 'bcde' 추출
'abcde'[ : 4] # 'abcde' 처음부터 4 번째 문자까지의 문자열 'abcd' 추출
'abcde'[ : -1] # 'abcde' 처음부터 끝에서부터 2 번째 문자까지의 문자열 'abcd' 추출
'abcde'[ : ] # 'abcde' 처음부터 마지막 문자까지의 문자열 'abcde' 추출
```

문자열 대소문자 변경

이 블록은 입력 문자열를 다음 형식 중 하나로 변환합니다:

- **대문자** (모든 문자 대문자로 변환)
- **소문자**
- **첫 글자만 대문자** (각 단어의 첫 글자만 대문자로 변환)

아래 블록의 결과는“HELLO”입니다.



Javascript 코드

```
function textToTitleCase(str) { // 첫 문자만 대문자로 변환하는 함수
    return str.replace(/\S+/g,
        function(txt) {return txt[0].toUpperCase() + txt.substring(1).toLowerCase();});
}

'hello'.toUpperCase(); // 문자열의 모든 문자를 대문자로 변환, 'HELLO' 반환
'hello'.toLowerCase(); // 문자열의 모든 문자를 소문자로 변환, 'hello' 반환
textToTitleCase('hello'); // 문자열의 각 단어를 대문자로 시작하고 나머지는 소문자로 변환, 'Hello' 반환
```

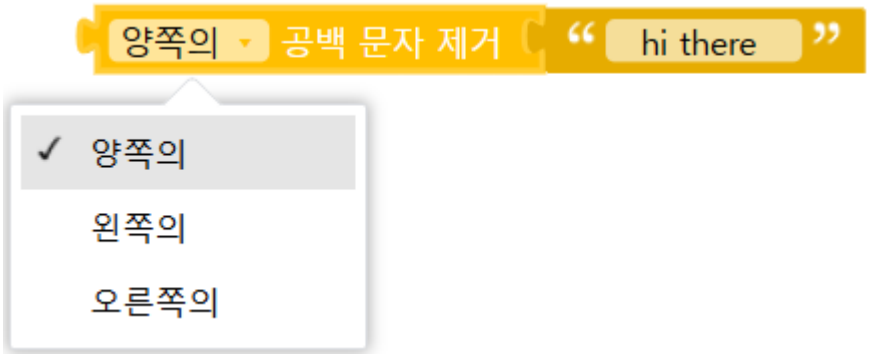
Python 코드

```
'hello'.upper() # 문자열의 모든 문자를 대문자로 변환, 'HELLO' 반환
'hello'.lower() # 문자열의 모든 문자를 소문자로 변환, 'hello' 반환
'hello'.title() # 문자열의 각 단어를 대문자로 시작하고 나머지는 소문자로 변환, 'Hello' 반환
```

공백 제거

다음 블록은 문자열에서 다음 위치의 공백을 제거합니다: - 문자열의 앞쪽 - 문자열의 끝쪽 - 양쪽 모두

아래 블록의 결과는“hi there”입니다. (중간의 공백은 그대로 유지됨)



Javascript 코드

```
' hi there '.trim(); // 양쪽의 공백 문자 제거, 'hi there' 반환
' hi there '.replace(/^[\s\xa0]+/, ''); // 왼쪽의 공백 문자 제거, 'hi there ' 반환
' hi there '.replace(/[\s\xa0]+$/, ''); // 오른쪽의 공백 문자 제거, ' hi there' 반환
```

Python 코드

```
' hi there '.strip() # 양쪽의 공백 문자 제거, 'hi there' 반환
' hi there '.lstrip() # 왼쪽의 공백 문자 제거, 'hi there ' 반환
' hi there '.rstrip() # 오른쪽의 공백 문자 제거, ' hi there' 반환
```

문자열에서 특정 문자열 숫자 세기

주어진 문자열에서 특정한 단어 (부분 문자열) 가 등장하는 횟수를 세어 반환하는 기능입니다.

Javascript 코드

```
function textCount(haystack, needle) { // 특정 문자열 집계 함수
    if (needle.length === 0) {
        return haystack.length + 1;
    } else {
        return haystack.split(needle).length - 1;
    }
}

textCount('abc, abc, ab', 'abc'); // "abc, abc, ab" 문자열에서 "abc" 의 갯수 2 반환
```

Python 코드

```
'abc, abc, ab'.count('abc') # "abc, abc, ab" 문자열에서 "abc" 의 갯수 2 반환
```

"abc, abc, ab" 에서 "abc" 와 일치하는 경우 2 를 반환합니다.

문자열에서 특정 문자 변경

문자열 내에서 특정 문자를 다른 문자로 변경하는 기능입니다. 원하는 문자를 선택하여 새로운 문자로 대체할 수 있습니다.

문자열 "abc, abc, ab" 에서 "abc" 를 "ABC" 로 변경합니다. 해당 블록의 실행 결과는 "ABC, ABC, ab" 입니다.

Javascript 코드

```
function textReplace(haystack, needle, replacement) { // 문자열에서의 특정 문자 변경 함수
    needle = needle.replace(/([-()\[\]{}+?*.$\^|,:#<!\\])/g, '\\$1').replace(/\x08/g, '\\x08');
    return haystack.replace(new RegExp(needle, 'g'), replacement);
}

textReplace('abc, abc, ab', 'abc', 'ABC'); // "abc, abc, ab" 문자열에서 "abc" 를 "ABC" 로 변환시 문자열 "ABC, ABC, ab" 반환
```

Python 코드

```
'abc, abc, ab'.replace('abc', 'ABC') # "abc, abc, ab" 문자열에서 "abc" 를 "ABC" 로 변환시 문자열 "ABC, ABC, ab" 반환
```

문자열 뒤집기

문자열의 순서를 거꾸로 바꾼 새로운 문자열을 반환하는 기능입니다.

“hello”를 뒤집은“olleh”를 반환합니다.

Javascript 코드

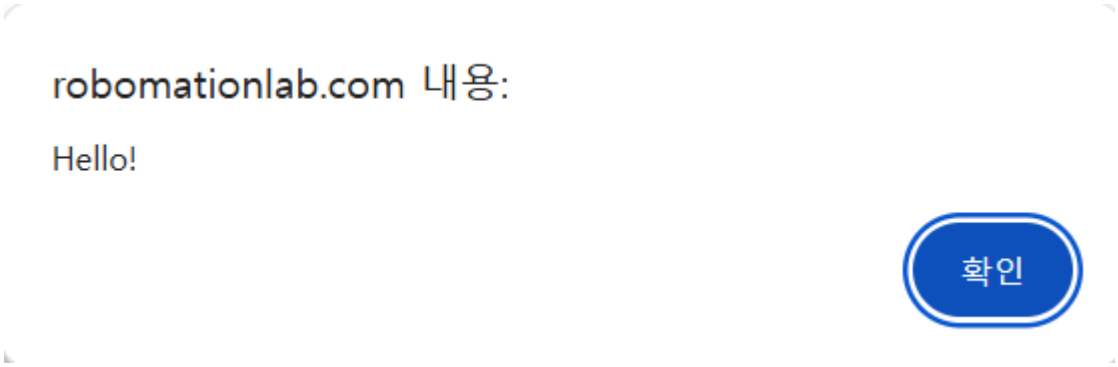
```
'hello'.split('').reverse().join('') // hello 문자열 뒤집은 결과인 "olleh" 반환
```

Python 코드

```
'hello'[::-1] # hello 문자열 뒤집은 결과인 "olleh" 반환
```

문자열 출력

출력 블록은 입력 값을 팝업 창에 표시합니다.



Javascript 코드

```
window.alert('Hello!'); // 팝업창에 "HELLO" 출력
```

Python 코드

```
print('Hello!') # 팝업창에 "HELLO" 출력
```

사용자 입력 받기

다음 블록은 사용자에게 이름 입력을 요청하는 팝업 창을 생성하며, 입력된 값은 **name** 변수에 저장됩니다.

Javascript 코드

```
var name2;
```

```
name2 = window.prompt('이름을 입력하세요:');
```

Python 코드

```
name = None

def text_prompt(msg): # 입력받는 함수
    try:
        return raw_input(msg)
    except NameError:
        return input(msg)

name = text_prompt('이름을 입력하세요:')
```

또한 사용자로부터 숫자를 입력받는 버전도 있습니다.

Javascript 코드

```
var age;

age = Number(window.prompt('나이을 입력하세요:'));
```

Python 코드

```
age = None

def text_prompt(msg): # 입력받는 함수
    try:
        return raw_input(msg)
    except NameError:
        return input(msg)
age = float(text_prompt('나이을 입력하세요:'))
```

리스트

리스트는“할 일 목록”이나“쇼핑 목록”처럼 **항목들이 순서대로 나열된 집합**입니다.
리스트의 항목들은 어떤 타입이라도 될 수 있으며, 동일한 값이 리스트에 여러 번 나타날 수 있습니다.

리스트 생성

빈 리스트 생성

가장 간단한 리스트는 빈 리스트로, **빈 리스트 생성** 블록을 사용하여 생성합니다:

Javascript 코드

```
[]; // 빈 리스트
```

Python 코드

```
[] # 빈 리스트
```

리스트 만들기

기본 사용법

리스트 만들기 블록을 사용하면 새로운 리스트에 초기값을 지정할 수 있습니다. 예를 들어, 단어 리스트를 만들고 이를 **letters** 라는 변수에 저장할 수 있습니다:

이 문서에서는 이 리스트를 [“하나”, “둘”, “셋”] 으로 나타냅니다. 이 블록에서 정의된 변수들은 이후 예제에서 사용될 것입니다.

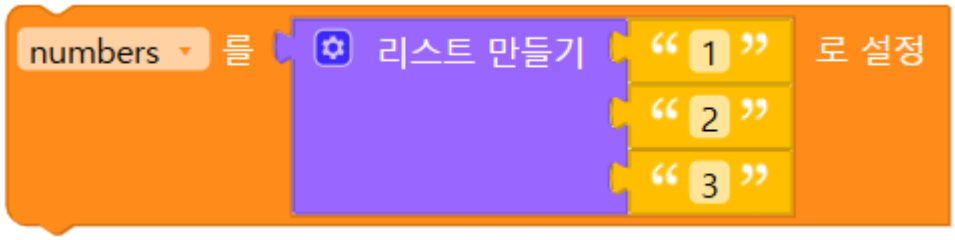
Javascript 코드

```
var letters;

letters = ['하나', '둘', '셋'];
```

Python 코드

```
letters = None

letters = ['하나', '둘', '셋']
```

다음은 숫자 형태의 문자열 리스트를 만드는 예입니다:



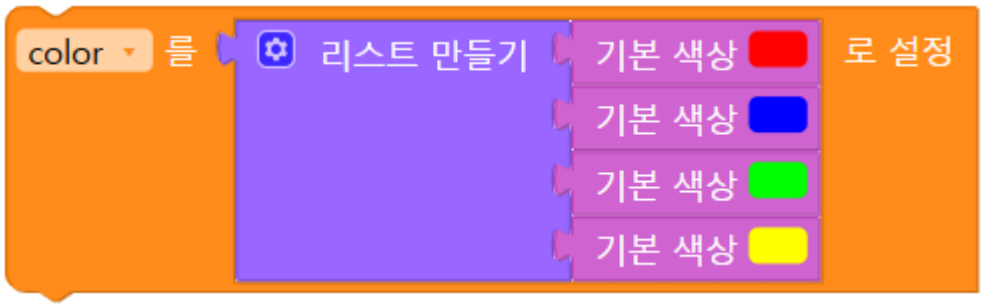
Javascript 코드

```
var numbers;

numbers = ['1', '2', '3'];
```

Python 코드

```
numbers = None

numbers = ['1', '2', '3']
```

다음은 색상 리스트를 만드는 예입니다:



Javascript 코드

```
color = [[255, 0, 0], [0, 0, 255], [0, 255, 0], [255, 255, 0]];
```

Python 코드

```
color = [[255, 0, 0], [0, 0, 255], [0, 255, 0], [255, 255, 0]]
```

덜 흔하지만, 다양한 타입의 값을 가진 리스트도 만들 수 있습니다:

Javascript 코드

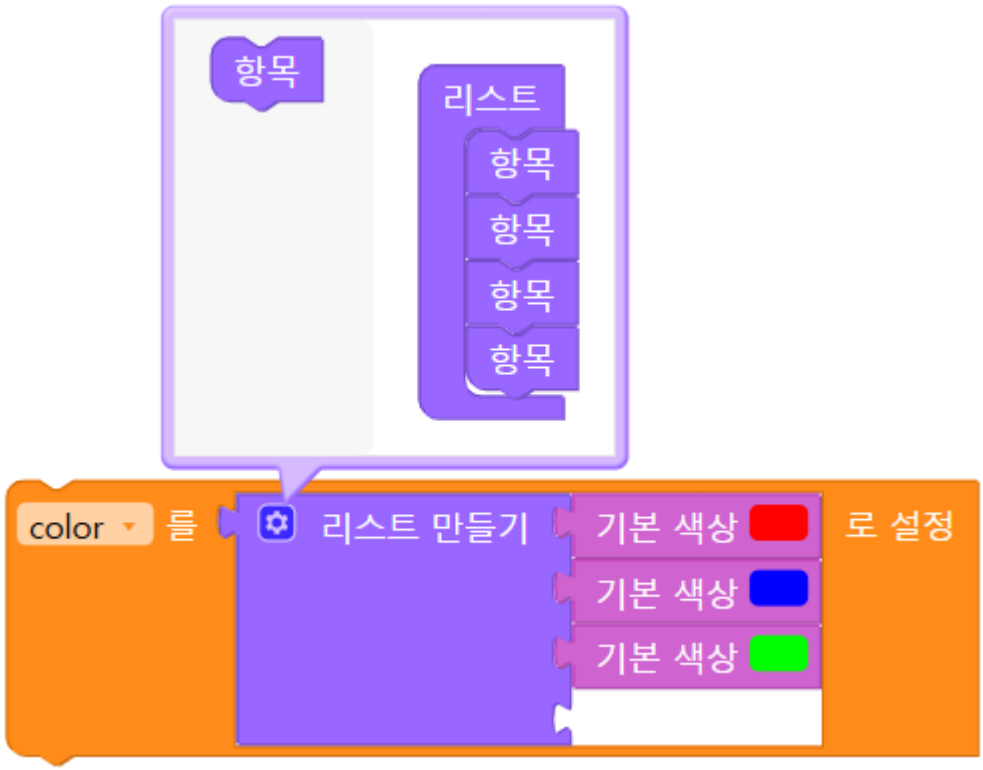
```
['하나', 1, [255, 0, 0]];
```

Python 코드

```
['하나', 1, [255, 0, 0]]
```

입력 항목 개수 변경

입력 항목의 개수를 변경하려면 기어 아이콘을 클릭하세요.
새로운 창이 열리며, 여기서 왼쪽의 **항목** 서브 블록을 오른쪽의 **리스트** 블록으로 드래그하여 새로운 입력을 추가할 수 있습니다:



새 항목은 원하는 위치에 추가할 수 있습니다. 마찬가지로, 원하지 않는 **항목** 서브 블록은 왼쪽으로 드래그하여 제거할 수 있습니다.

항목으로 리스트 설정

항목으로 리스트 설정 블록을 사용하면 지정된 항목을 반복하여 원하는 개수만큼 리스트를 만들 수 있습니다.
예를 들어, 아래의 블록은 변수 **words** 를 [“very”, “very”, “very”] 로 설정합니다.

Javascript 코드

```
var words;

function listsRepeat(value, n) { // 리스트 내 동일 값 반복 생성 함수
    var array = [];
    for (var i = 0; i < n; i++) {
        array[i] = value;
    }
```

```
    return array;
}

words = listsRepeat('very', 3); // words = ['very', 'very', 'very']
```

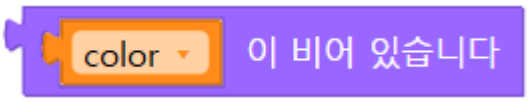
Python 코드

```
words = None
words = ['very'] * 3 # words = ['very', 'very', 'very']
```

리스트 길이

비어 있습니다

비어 있습니다 블록의 값은 입력이 빈 리스트일 경우 **참**, 그 외에는 **거짓**입니다. (리스트가 아니더라도 마찬가지)
아래 블록의 값은 **거짓**이 됩니다. 왜냐하면 **color** 변수는 비어 있지 않으며, 3 개의 항목을 가지고 있기 때문입니다.



Javascript 코드

```
var color;

color = [[255, 0, 0], [0, 0, 255], [0, 255, 0]];
!color.length; // 비어있지 않으므로 거짓 출력
```

Python 코드

```
color = None

color = [[255, 0, 0], [0, 0, 255], [0, 255, 0]]
not len(color) # 비어있지 않으므로 거짓 출력
```

이것은 문자열의“빈 문자열 확인”블록과 유사합니다.

길이

길이 블록의 값은 리스트의 항목 수입니다.

예를 들어, 아래 블록에서 **color** 는 3 개의 항목을 가졌기 때문에 3 을 반환합니다.

Javascript 코드

```
var color;

color = [[255, 0, 0], [0, 0, 255], [0, 255, 0]];
color.length; // 3 개의 항목이므로 3 의 값 반환
```

Python 코드

```
color = None

color = [[255, 0, 0], [0, 0, 255], [0, 255, 0]]
len(color)  # 3 개의 항목이므로 3 의 값 반환
```

길이 블록은 리스트에 몇 개의 항목이 있는지 알려줍니다.

리스트 내에 다른 항목이 몇 개 있는지가 아니라, 그 자체의 항목 수를 세는 것입니다.

예를 들어, 아래 **words** 는“very”가 세 번 반복된 리스트 ([“very”, “very”, “very”]) 입니다.

이때 아래 블록은 3 을 반환합니다.,

이것은 문자열의“문자열 길이”블록과 유사합니다.

Javascript 코드

```
var words;

function listsRepeat(value, n) { // 리스트 내 동일 값 반복 생성 함수
```

```
    var array = [];
    for (var i = 0; i < n; i++) {
        array[i] = value;
    }
    return array;
}

words = listsRepeat('very', 3);
words.length; // 3 개의 항목이므로 3 의 값 반환
```

Python 코드

```
words = None

words = ['very'] * 3
len(words) # 3 개의 항목이므로 3 의 값 반환
```

리스트 나타난 위치

이 블록들은 리스트에서 항목의 위치를 찾습니다.
예를 들어, 아래 블록의 값은 1 입니다. “very”라는 첫 번째 항목이 **words** 리스트 ([“very”, “very”, “very”]) 에서 첫 번째 위치에 있기 때문입니다.

다음 결과는 3 입니다. “very”라는 항목이 **words** 리스트에서 마지막 위치에 있기 때문입니다.

리스트에 항목이 없는 경우, 결과는 0 입니다.

이 블록들은 문자열의“문자열 찾기”블록과 유사합니다.

Javascript 코드

```javascript
var words;

function listsRepeat(value, n) { // 리스트 내 동일 값 반복 생성 함수
    var array = [];
    for (var i = 0; i < n; i++) {
        array[i] = value;
    }
    return array;
}

words = listsRepeat('very', 3);
words.indexOf('very') + 1 // "very" 문자열이 첫 번째로 나타난 위치 1 반환
words.lastIndexOf('very') + 1 // "very" 문자열이 마지막으로 나타난 위치 3 반환
words.indexOf('유니콘') + 1 // " 유니콘" 문자열이 첫 번째로 나타난 위치 0 반환
```

Python 코드

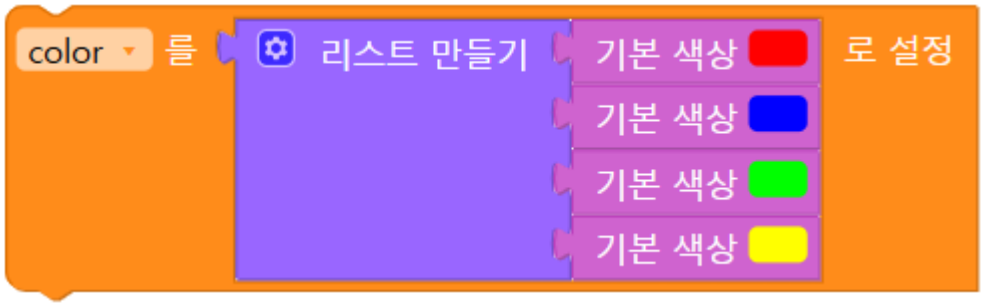
```python
words = None

def first_index(my_list, elem): # 문자열 첫 번째 위치 찾기 함수
    try: index = my_list.index(elem) + 1
    except: index = 0
    return index

def last_index(my_list, elem): # 문자열 마지막 위치 찾기 함수
    try: index = len(my_list) - my_list[::-1].index(elem)
    except: index = 0
    return index

words = ['very'] * 3
first_index(words, 'very') # "very" 문자열이 첫 번째로 나타난 위치 1 반환
last_index(words, 'very') # "very" 문자열이 마지막으로 나타난 위치 3 반환
```

```
first_index(words, '유니콘') # " 유니콘" 문자열이 첫 번째로 나타난 위치 0 반환
```

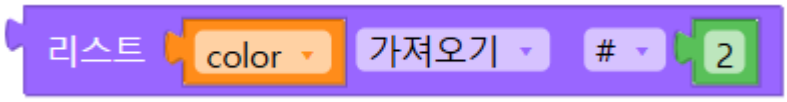
리스트에서 항목 가져오기

단일 항목 가져오기

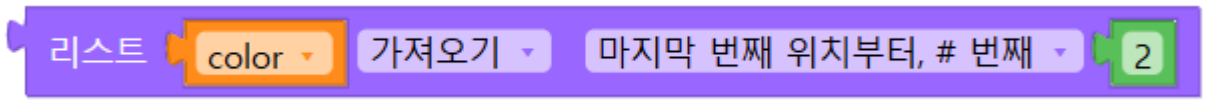
color 리스트의 정의를 기억하세요:



다음 블록은 색상 파랑을 가져옵니다. 파랑은 리스트의 두 번째 항목이기 때문입니다 (왼쪽에서부터 세기):

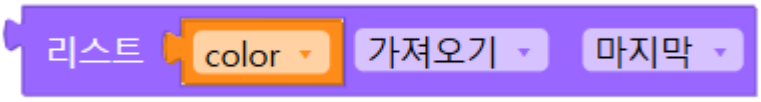


이 블록은 초록을 가져옵니다. 초록은 오른쪽 끝에서 두 번째 항목이기 때문입니다:



이 블록은 첫 번째 항목인 빨강을 가져옵니다:

이 블록은 마지막 항목인 노랑을 가져옵니다:



이 블록은 리스트에서 임의로 하나의 항목을 선택합니다. 빨강, 파랑, 초록, 노랑 중 하나를 동일한 확률로 선택합니다.

Javascript 코드

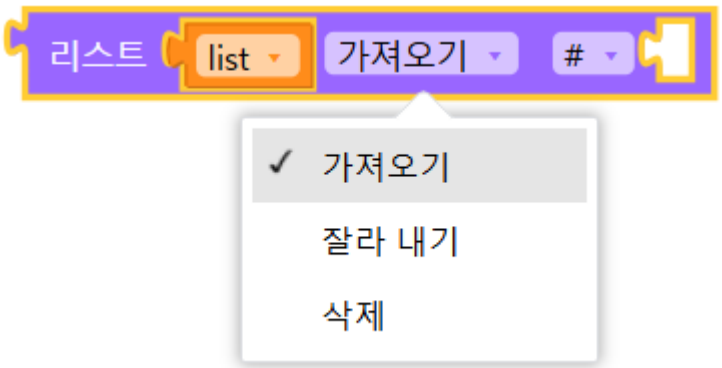
```javascript
var color;

function listsGetRandomItem(list, remove) { // 리스트 무작위 항목 선택 함수
    var x = Math.floor(Math.random() * list.length);
    if (remove) {
        return list.splice(x, 1)[0];
    } else {
        return list[x];
    }
}

color = [[255, 0, 0], [0, 0, 255], [0, 255, 0], [255, 255, 0]];
color[1]; // color 에서 두 번째 항목 파랑 가져오기
color.slice(-2)[0]; // color 에서 끝에서 두 번째 항목 초록 가져오기
color[0]; // color 에서 첫 번째 항목 빨강 가져오기
color.slice(-1)[0]; // color 에서 마지막 항목 노랑 가져오기
listsGetRandomItem(color, false); // color 에서 무작위 항목 가져오기
```

Python 코드

```python
import random

color = None

color = [[255, 0, 0], [0, 0, 255], [0, 255, 0], [255, 255, 0]]
color[1] # color 에서 두 번째 항목 파랑 가져오기
color[-2] # color 에서 끝에서 두 번째 항목 초록 가져오기
color[0] # color 에서 첫 번째 항목 빨강 가져오기
color[-1] # color 에서 마지막 항목 노랑 가져오기
random.choice(color) # color 에서 무작위 항목 가져오기
```

항목 가져오기 및 제거

리스트에서…가져오기 블록에서 드롭다운 메뉴를 선택하면 **리스트에서…가져오기 및 제거**로 변경되어, 항목을 가져오는 것뿐만 아니라 원본 리스트도 수정합니다:



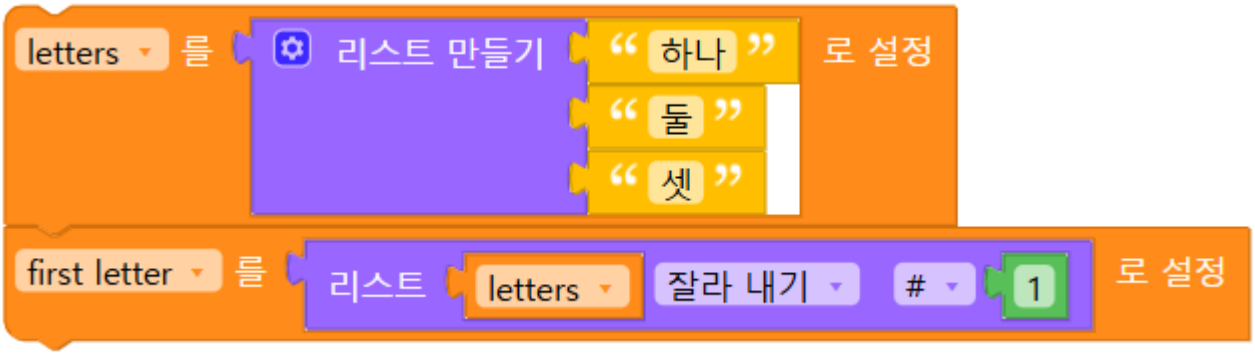
Javascript 코드

```
var list;

function listsGetRandomItem(list, remove) { // 리스트 랜덤 항목 추출 함수
    var x = Math.floor(Math.random() * list.length);
    if (remove) {
        return list.splice(x, 1)[0];
    } else {
        return list[x];
    }
}

list[0]; // 리스트 첫 번째 항목 가져오기
list.slice(-1)[0]; // 리스트 마지막 항목 가져오기
listsGetRandomItem(list, false); // 리스트 랜덤 항목 가져오기
list.splice(0, 1)[0]; // 리스트 첫 번째 항목 잘라내기
list.shift(); // 리스트 앞에서 항목 제거 (첫 번째 항목 반환)
list.pop(); // 리스트 마지막 항목 잘라내기
listsGetRandomItem(list, true); // 리스트 임의 항목 잘라내기
```

Python 코드

```
list2 = None # list 는 예약어 이기 때문에 list2 사용

def lists_remove_random_item(myList): # 리스트 임의 잘라내기 함수
    x = int(random.random() * len(myList))
    return myList.pop(x)



list2[0] # 리스트 첫 번째 항목 가져오기
list2[-1] # 리스트 마지막 항목 가져오기
random.choice(list2) # 리스트 랜덤 항목 가져오기
list2.pop(0) # 리스트 첫 번째 항목 잘라내기
list2.pop(0) # 리스트 앞에서 항목 제거 (첫 번째 항목 반환)
list2.pop() # 리스트 마지막 항목 잘라내기
lists_remove_random_item(list2) # 리스트 임의 항목 잘라내기
```

이 예제는 변수 **first letter** 에 [“하나”] 를 설정하고, **letters** 는 [“둘”, “셋”] 으로 남겨둡니다.



Javascript 코드

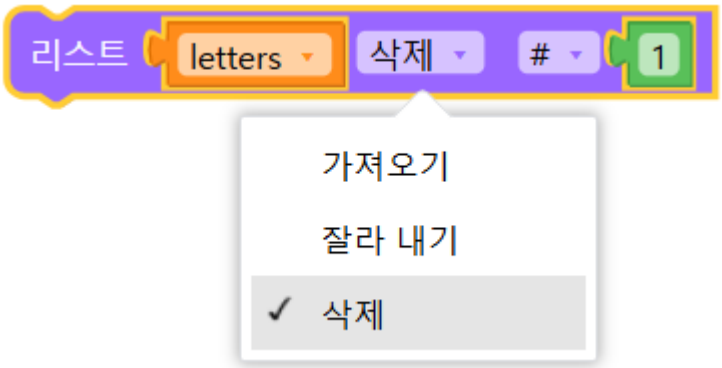
```
var letters, first_letter;

letters = ['하나', '둘', '셋'];
first_letter = letters.splice(0, 1)[0]; // letters 리스트 첫 번째 항목 잘라내기 first_letter = ["
하나], letters = [" 둘", " 셋"]
```

Python 코드

```
letters = None
first_letter = None

letters = ['하나', '둘', '셋']
first_letter = letters.pop(0) # letters 리스트 첫 번째 항목 잘라내기 first_letter = [" 하나],
letters = [" 둘", " 셋"]
```

항목 제거

드롭다운에서“삭제”를 선택하면 블록 왼쪽의 플러그가 사라집니다:



이렇게 하면 **letters** 에서 첫 번째 항목이 제거됩니다.

Javascript 코드

```
var letters;

letters = ['하나', '둘', '셋'];
letters.splice(0, 1); // letters 리스트 첫 번째 항목 삭제, letters = [" 둘"," 셋"]
```

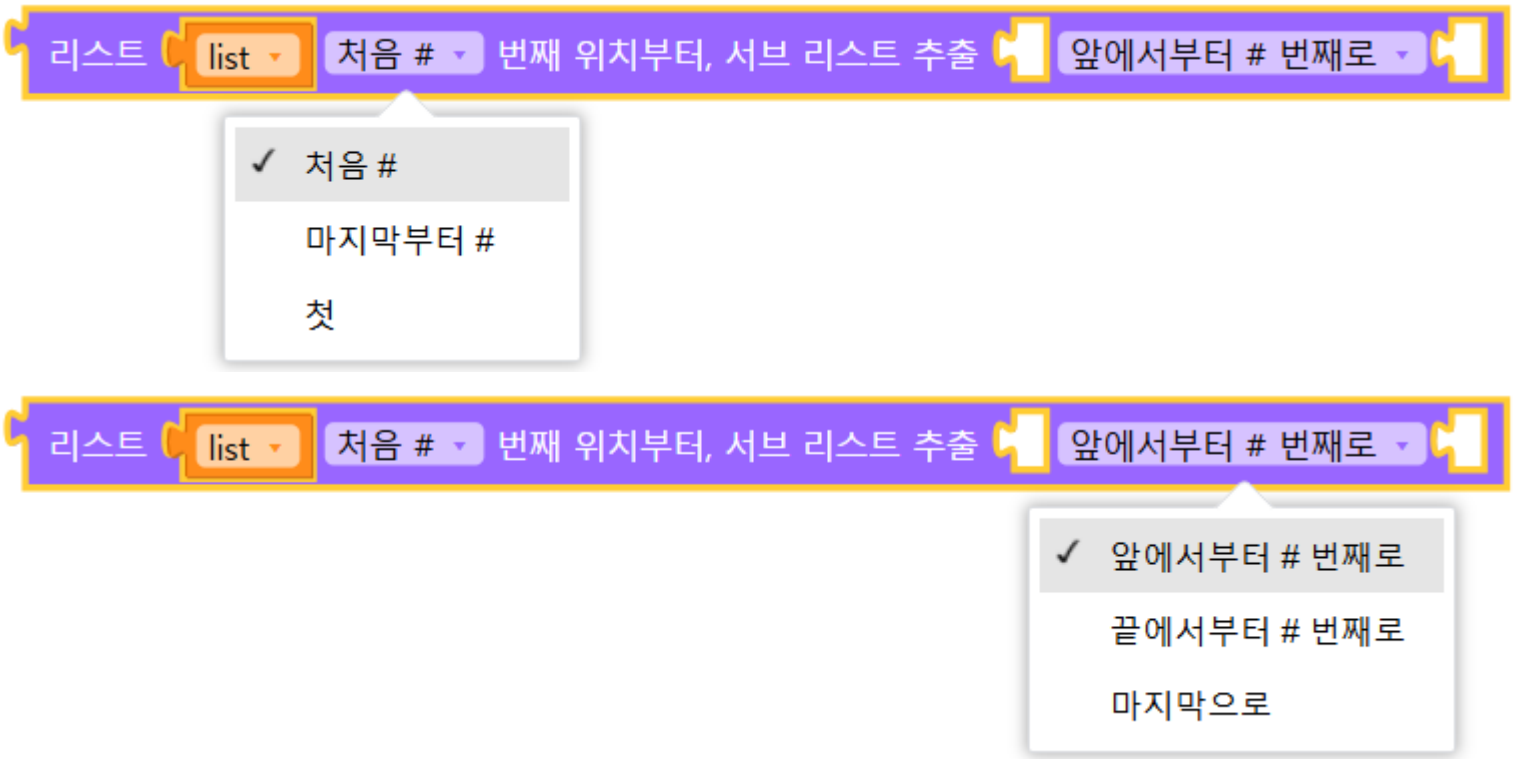
Python 코드

```
letters = None

letters = ['하나', '둘', '셋']
letters.pop(0) # letters 리스트 첫 번째 항목 삭제, letters = [" 둘"," 셋"]
```

서브리스트 추출

리스트에서…서브리스트 추출 블록은 **리스트에서…가져오기** 블록과 유사하지만, 개별 항목 대신 서브리스트를 추출합니다.

서브리스트의 시작과 끝을 지정하는 방법은 여러 가지가 있습니다:



Javascript 코드

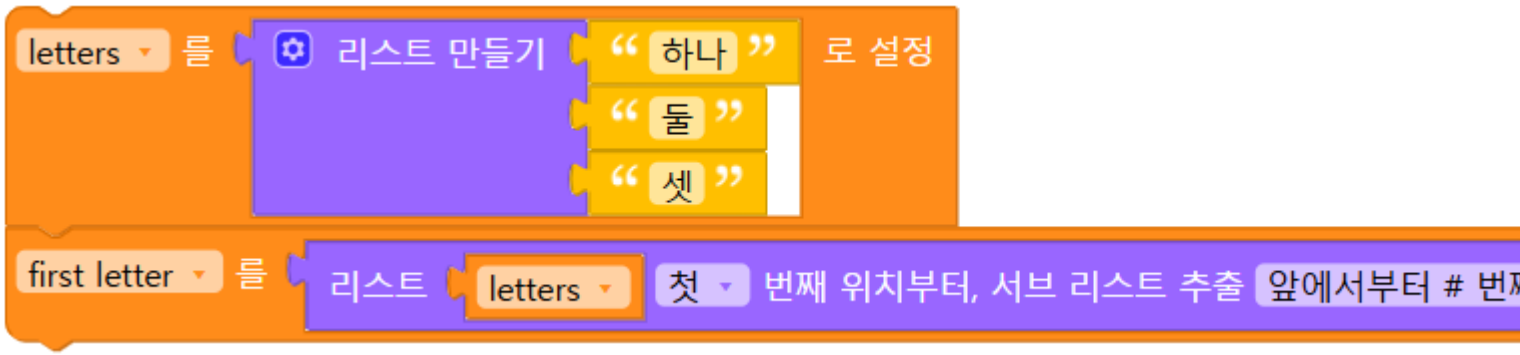
```
var list;

list.slice(0, 1); // 첫 번째 위치부터 첫 번째 항목까지 서브리스트 추출
list.slice(0, list.length - 0); // 첫 번째 위치부터 끝에서 첫 번째까지 서브리스트 추출
list.slice(0, list.length); // 첫 번째 위치부터 마지막까지 서브리스트 추출
list.slice(list.length - 1, 1); // 마지막 첫 번째 위치부터 마지막까지 서브리스트 추출
list.slice(list.length - 1, list.length - 0); // 마지막 첫 번째 위치부터 마지막까지 서브리스트 추출
list.slice(list.length - 1, list.length); // 마지막 첫 번째 위치부터 마지막까지 서브리스트 추출
list.slice(0, 1); // 첫 번째 위치부터 첫 번째 항목까지 서브리스트 추출
list.slice(0, list.length - 0); // 첫 번째 위치부터 끝까지 서브리스트 추출
list.slice(0); // 첫 번째 위치부터 마지막까지 서브리스트 추출
```

Python 코드

```
list2 = None # list 는 예약어 이기 때문에 list2 사용

list2[ : 1] # 첫 번째 위치부터 첫 번째 항목까지 서브리스트 추출
list2[ : ] # 첫 번째 위치부터 끝까지 서브리스트 추출
list2[ : ] # 첫 번째 위치부터 마지막까지 서브리스트 추출
list2[-1 : 1] # 마지막 첫 번째 위치부터 끝까지 서브리스트 추출
list2[-1 : ] # 마지막 첫 번째 위치부터 끝까지 서브리스트 추출
list2[-1 : ] # 마지막 첫 번째 위치부터 마지막까지 서브리스트 추출
list2[ : 1] # 첫 번째 위치부터 첫 번째 항목까지 서브리스트 추출
list2[ : ] # 첫 번째 위치부터 끝까지 서브리스트 추출
list2[ : ] # 첫 번째 위치부터 마지막까지 서브리스트 추출
```

이 예제에서는 새로운 리스트 **first letters** 가 생성됩니다.

이 리스트에는 두 개의 항목이 포함됩니다: [“하나”, “둘”].



이 블록은 원본 리스트를 수정하지 않음을 유의하세요.

Javascript 코드

```
var letters, first_letter, list;

letters = ['하나', '둘', '셋'];
first_letter = list.slice(0, 2); // 첫 번째 위치부터 두 번째 항목까지 추출 first_letter = [" 하나",
" 둘"]
```

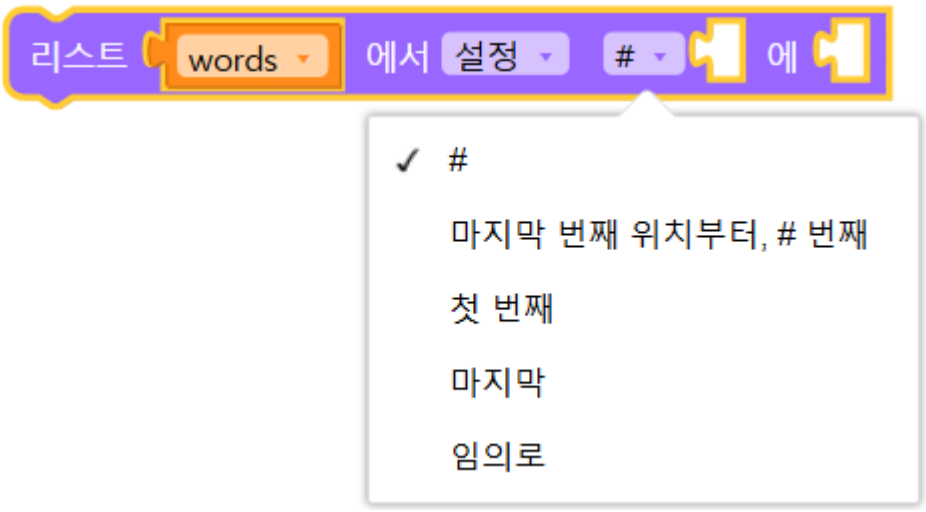
Python 코드

```
letters = None
first_letter = None
list2 = None # list 는 예약어 이기 때문에 list2 사용

letters = ['하나', '둘', '셋']
first_letter = list2[ : 2] # 첫 번째 위치부터 두 번째 항목까지 추출 first_letter = [" 하나", " 둘"]
```

리스트에 항목 추가하기

리스트에서…설정

리스트에서…설정 블록은 지정된 위치에 있는 항목을 다른 항목으로 교체합니다.



Javascript 코드

```
var letters, list;

list[0] = 'hello'; // 리스트에 첫 번째 위치에 "hello" 문자열 추가하기
list[list.length - 1] = 'hello'; // 리스트에 마지막 위치부터 "hello" 문자열 추가하기
list[0] = 'hello'; // 리스트 첫 번째에 "hello" 문자열 추가하기
list[list.length - 1] = 'hello'; // 리스트에 마지막 위치에 "hello" 문자열 추가하기
var tmpX = Math.floor(Math.random() * list.length); // 무작위 위치 계산
list[tmpX] = 'hello'; // 리스트에 무작위 위치에 "hello" 문자열 추가하기
```

Python 코드

```
import random

letters = None
list2 = None # list 는 예약어 이기 때문에 list2 사용

list2[0] = 'hello' # 리스트에 첫 번째 위치에 "hello" 문자열 추가하기
list2[-1] = 'hello' # 리스트에 마지막 위치부터 "hello" 문자열 추가하기
list2[0] = 'hello' # 리스트 첫 번째에 "hello" 문자열 추가하기
list2[-1] = 'hello' # 리스트의 마지막 위치에 "hello" 문자열 추가하기
tmp_x = int(random.random() * len(list2))  # 무작위 위치 계산
list2[tmp_x] = 'hello' # 리스트에 무작위 위치에 "hello" 문자열 추가하기
```

드롭다운 옵션의 의미는이전 섹션을 참조하세요.

다음 예제는 두 가지 일을 합니다: 1. **words** 리스트를 [“very”, “very”, “very”] 로 생성합니다. 2. 리스트의 세 번째 항목을“good”으로 교체합니다. 새 값은 **words** 가 [“very”, “very”, “good”] 이 됩니다.

Javascript 코드

```
var words;

function listsRepeat(value, n) { // 리스트 내 동일 값 반복 생성 함수
    var array = [];
    for (var i = 0; i < n; i++) {
        array[i] = value;
    }
    return array;
}
```

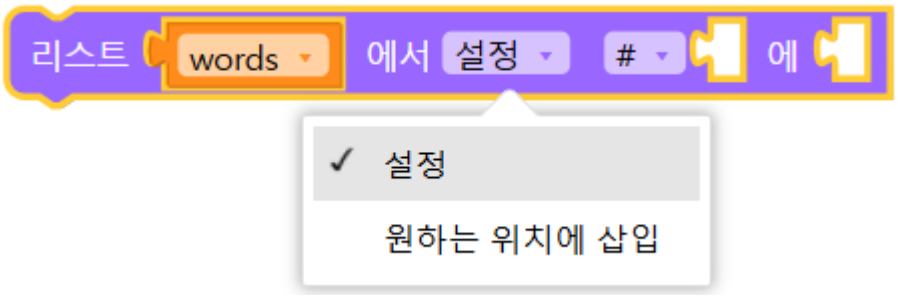
```
words = listsRepeat('very', 3); // words = ["very","very","very"]
words[2] = 'good';// words = ["very","very","good"]
```

Python 코드

```
words = None

words = ['very'] * 3 # words = ['very','very','very']
words[2] = 'good'# words = ['very','very','good']
```

리스트에서…원하는 위치에 삽입

리스트에서…원하는 위치에 삽입 블록은 **리스트에서…설정** 블록에서 드롭다운 메뉴를 사용하여 얻을 수 있습니다:



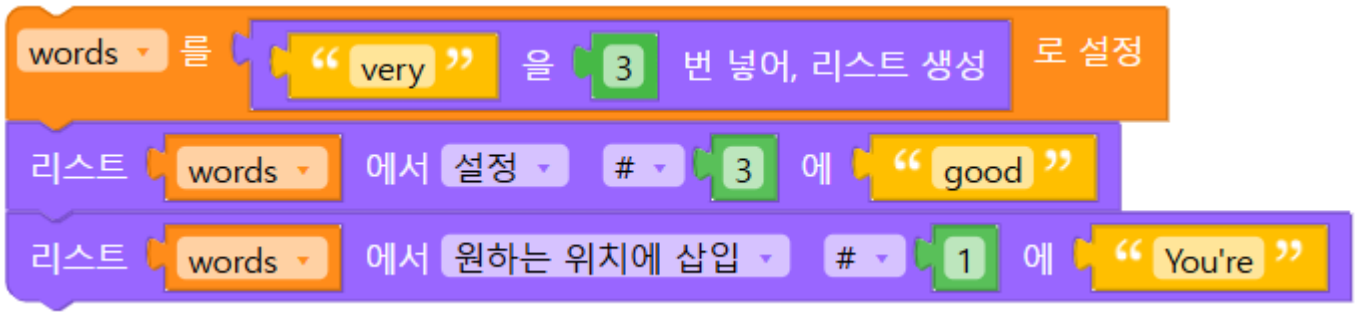
Javascript 코드

```
var words;

words.splice(2, 0, 'good'); // 세번째 위치 앞에 "good" 문자열 삽입
words.splice(words.length - 3, 0, 'good'); // 마지막으로부터 3 번째 위치 앞에 "good" 문자열 삽입
words.unshift('good'); // 맨 앞에 "good" 문자열 삽입
words.push('good'); // 마지막 위치에 "good" 문자열 삽입
var tmpX = Math.floor(Math.random() * words.length); // 무작위 위치 계산
words.splice(tmpX, 0, 'good'); // 무작위 위치에 "good" 문자열 삽입
```

Python 코드

```
import random

words = None

words.insert(2, 'good') # 세번째 위치 앞에 "good" 문자열 삽입
words.insert(-3, 'good') # 마지막으로부터 3 번째 위치 앞에 "good" 문자열 삽입
words.insert(0, 'good') # 맨 앞에 "good" 문자열 삽입
words.append('good') # 마지막 위치에 "good" 문자열 삽입
tmp_x = int(random.random() * len(words)) # 무작위 위치 계산
words.insert(tmp_x, 'good') # 무작위 위치에 "good" 문자열 삽입
```

이 블록은 새 항목을 리스트의 지정된 위치에 삽입합니다. 예를 들어, 아래 예제에서는 세 가지 일이 일어납니다: 1. **words** 리스트는 [“very”, “very”, “very”] 로 생성됩니다. 2. 리스트의 세 번째 항목이“good”으로 교체됩니다. 새로운 값은 [“very”, “very”, “good”] 입니다. 3. “you’re”라는 단어가 리스트의 맨 앞에 삽입됩니다. 최종 값은 [“You’re” , “very”, “very”, “good”] 입니다.



Javascript 코드

```
var words;

function listsRepeat(value, n) { // 리스트 내 동일 값 반복 생성 함수
    var array = [];
    for (var i = 0; i < n; i++) {
        array[i] = value;
    }
    return array;
}
```

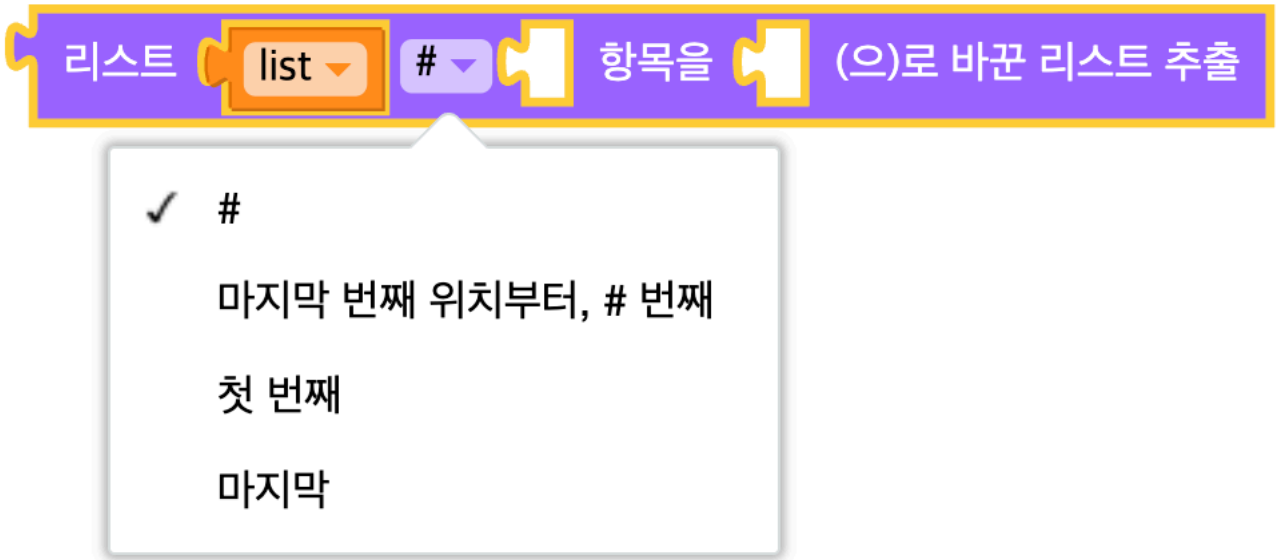
```
words = listsRepeat('very', 3);
words[2] = 'good'; // words = ["very", "very", "good"]
words.splice(0, 0, 'You\'re'); // words = ["You're", "very", "very", "good"]
```

Python 코드

```
words = None

words = ['very'] * 3
words[2] = 'good' # words = ['very', 'very', 'good']
words.insert(0, "You're") # words = ['You're', 'very', 'very', 'good']
```

리스트 값 치환 및 리스트 반환

리스트…항목을…(으) 로 바꾼 리스트 추출 블록은 지정된 위치에 있는 항목을 다른 항목으로 치환한 뒤, 전체 리스트를 반환합니다.



Javascript 코드

```
var words;

words.map((data, idx) => (idx === 0 ? 'good' : data)) // words 리스트의 첫번째 항목을 "good" 으로 바꾼 리스트 추출
```

```
words.map((data, idx) => (idx === words.length - 1 ? 'good' : data)) // words 리스트의 마지막으로부터 첫번째 항목을 "good" 으로 바꾼 리스트 추출
words.map((data, idx) => (idx === 0 ? 'good' : data))// words 리스트의 첫번째 항목을 "good" 으로 바꾼 리스트 추출
words.map((data, idx) => (idx === words.length - 1 ? 'good' : data))// words 리스트의 마지막 항목을 "good" 으로 바꾼 리스트 추출
```

Python 코드

```
words = None

['good' if i == 0 else data for i, data in enumerate(words)] # words 리스트의 첫번째 항목을 "good" 으로 바꾼 리스트 추출
['good' if i == len(words) - 1 else data for i, data in enumerate(words)] # words 리스트의 마지막으로부터 첫번째 항목을 "good" 으로 바꾼 리스트 추출
['good' if i == 0 else data for i, data in enumerate(words)] # words 리스트의 첫번째 항목을 "good" 으로 바꾼 리스트 추출
['good' if i == len(words) - 1 else data for i, data in enumerate(words)] # words 리스트의 마지막 항목을 "good" 으로 바꾼 리스트 추출
```

드롭다운 옵션의 의미는이전 섹션을 참조하세요.

다음 예제는 세 가지 일을 합니다:

1. **words** 리스트를 [“very”, “very”, “very”] 로 생성합니다.
2. 리스트의 세 번째 항목을“good”으로 교체한 새로운 리스트를 반환하고, 해당 리스트를 새로운 변수 **list** 에 할당합니다.
3. **list** 리스트는 [“very”, “very”, “good”] 이 됩니다. **words** 리스트의 값은 바뀌지 않습니다.

Javascript 코드

```javascript
var words, list;

function listsRepeat(value, n) { // 리스트 내 동일 값 반복 생성 함수
    var array = [];
    for (var i = 0; i < n; i++) {
        array[i] = value;
    }
    return array;
}

words = listsRepeat('very', 3); // words = ["very", "very", "very"]
list = (list.map((data, idx) => (idx === 2 ? 'good' : data)));// list = ["very", "very", "good"]
```

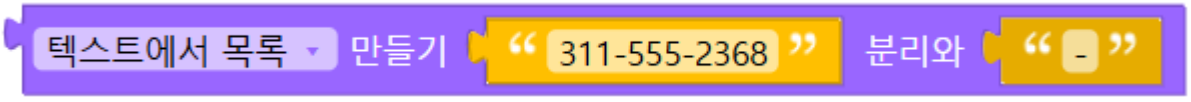
Python 코드

```python
words = None
list2 = None # list 는 예약어 이기 때문에 list2 사용

words = ['very'] * 3 # words = ['very', 'very', 'very']
list2 = ['good' if i == 2 else data for i, data in enumerate(list2)] # list2 = ['very', 'very',
'good']
```

문자열 분할 및 리스트 결합

텍스트에서 목록 만들기

텍스트에서 목록 만들기 블록은 주어진 텍스트를 구분자를 사용하여 조각으로 나눕니다:



Javascript 코드

```
'311-555-2368'.split('-'); // '-' 를 기준으로 분리, ["311", "555", "2368"]
```

Python 코드

```
'311-555-2368'.split('-') # '-' 를 기준으로 분리, ["311", "555", "2368"]
```

위 예제에서 새 리스트는“311”, “555”, “2368”로 나뉩니다.

목록에서 텍스트 만들기

목록에서 텍스트 만들기 블록은 구분자를 사용하여 리스트의 항목들을 하나의 텍스트로 결합합니다:

Javascript 코드

```
['311', '555', '2368'].join('-'); // '-'를 구분자로 사용하여 리스트를 하나의 문자열로 결합
"311-555-2368"
```

Python 코드

```
'-'.join(['311', '555', '2368']) # '-'를 구분자로 사용하여 리스트를 하나의 문자열로 결합
"311-555-2368"
```

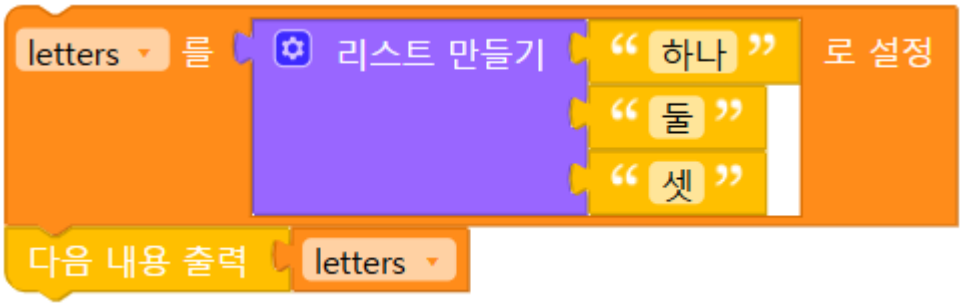
위 예제에서 새 텍스트는“311-555-2368”로 반환됩니다.

관련 블록

리스트 출력하기

텍스트의“출력”블록은 리스트를 출력할 수 있습니다.

아래 프로그램의 결과는 알림 상자에 출력됩니다:



Javascript 코드

```
var letters;

letters = ['하나', '둘', '셋'];
window.alert(letters); // 리스트 출력하기
```

Python 코드

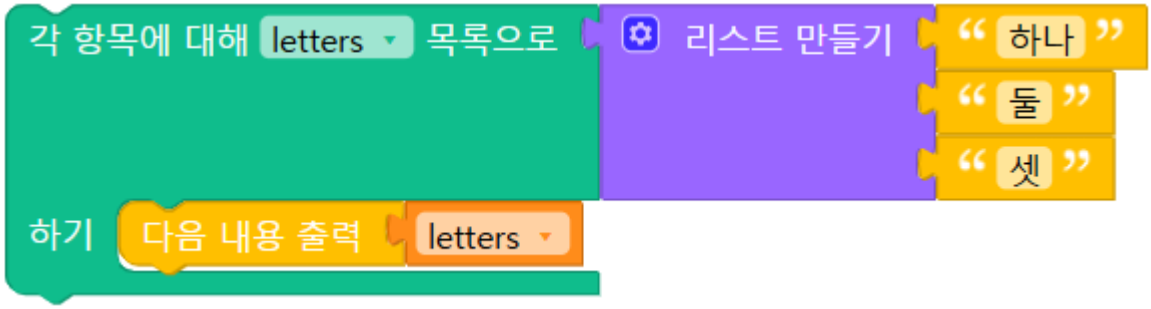
```
letters = None

letters = ['하나', '둘', '셋']
print(letters) # 리스트 출력하기
```

하나, 둘, 셋이 출력됩니다.

리스트의 각 항목에 대해

반복의"각 항목에 대해"블록은 리스트의 각 항목에 대해 연산을 수행합니다.

예를 들어, 아래 블록들은 리스트의 각 항목을 개별적으로 출력합니다:



Javascript 코드

```javascript
var letters;

var letters_list = ['하나', '둘', '셋'];
for (var letters_index in letters_list) { // 리스트 각 항목에 대해 출력하기
    letters = letters_list[letters_index];
    window.alert(letters);
}
```

Python 코드

```python
letters = None
for letters in ['하나', '둘', '셋']: # 리스트 각 항목에 대해 출력하기
    print(letters)
```

하나, 둘, 셋이 순차적으로 출력됩니다.
이 작업은 원본 리스트에서 항목을 제거하지 않습니다.

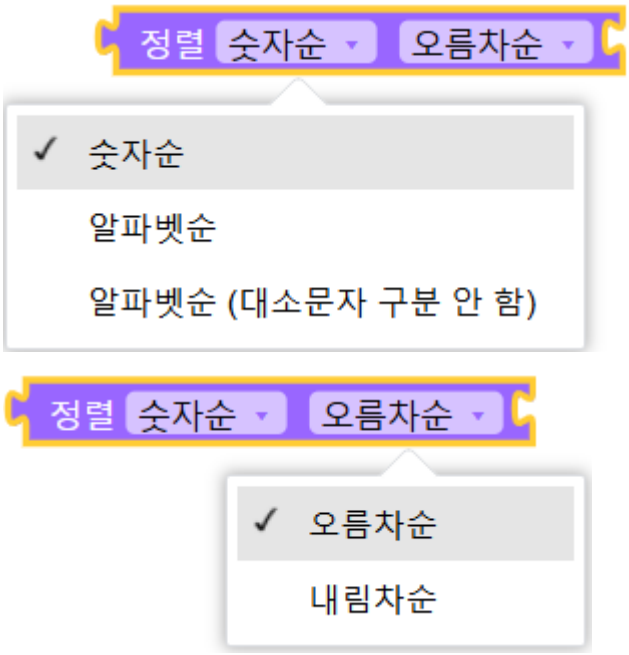
반복 종료 블록들의 예시도 참고하세요.

리스트 순서

리스트 항목의 순서를 변경할 수 있는 블록입니다.

정렬

리스트를 원하는 기준에 따라 **정렬하는 블록**입니다.
숫자 또는 알파벳을 기준으로 정렬할 수 있으며, 오름차순과 내림차순을 선택할 수 있습니다.
또한, 알파벳 정렬 시 대소문자를 무시하고 정렬할 수도 있습니다.



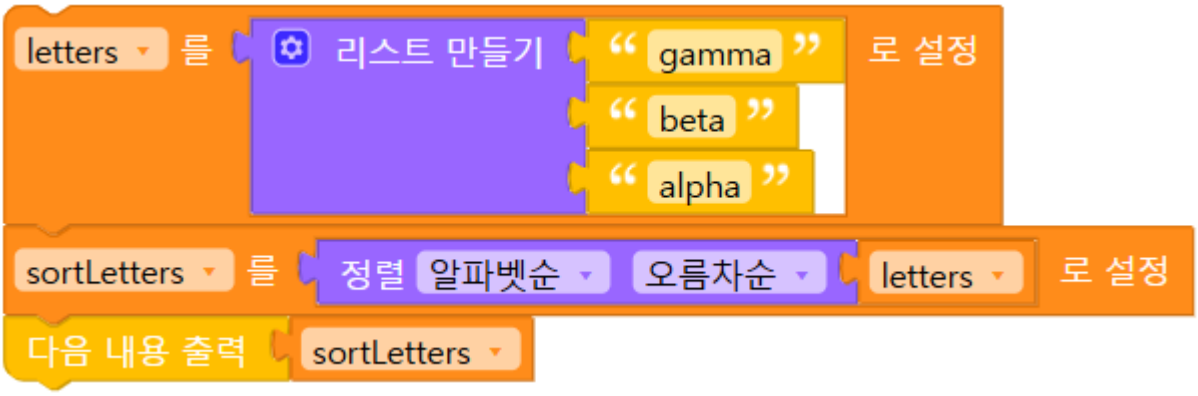
Javascript 코드

```
function listsGetSortCompare(type, direction) { // 문자 비교 함수
    var compareFuncs = {
        'NUMERIC': function(a, b) {
                return Number(a) - Number(b); },
        'TEXT': function(a, b) {
                return String(a) > String(b) ? 1 : -1; },
        'IGNORE_CASE': function(a, b) {
                return String(a).toLowerCase() > String(b).toLowerCase() ? 1 : -1; },
    };
    var compare = compareFuncs[type];
    return function(a, b) { return compare(a, b) * direction; };
}

[].slice().sort(listsGetSortCompare("NUMERIC", 1)); // 숫자, 오름차순 정렬
[].slice().sort(listsGetSortCompare("NUMERIC", -1)); // 숫자, 내림차순 정렬
[].slice().sort(listsGetSortCompare("TEXT", 1)); // 문자, 오름차순 정렬
[].slice().sort(listsGetSortCompare("TEXT", -1)) // 문자, 내림차순 정렬
[].slice().sort(listsGetSortCompare("IGNORE_CASE", 1)) // 알파벳순 (대소문자 구별 X), 오름차순 정렬
[].slice().sort(listsGetSortCompare("IGNORE_CASE", -1)) // 알파벳순 (대소문자 구별 X), 내림차순 정렬
```

Python 코드

```
def lists_sort(my_list, type, reverse): # 정렬 함수
    def try_float(s):
        try:
            return float(s)
        except:
            return 0
    key_funcs = {
        "NUMERIC": try_float,
        "TEXT": str,
        "IGNORE_CASE": lambda s: str(s).lower()
    }
    key_func = key_funcs[type]
    list_cpy = list(my_list)
    return sorted(list_cpy, key=key_func, reverse=reverse)

lists_sort([], "NUMERIC", False) # 숫자, 오름차순 정렬
lists_sort([], "NUMERIC", True) # 숫자, 내림차순 정렬
lists_sort([], "TEXT", False) # 문자, 오름차순 정렬
lists_sort([], "TEXT", True) # 문자, 내림차순 정렬
lists_sort([], "IGNORE_CASE", False) # 알파벳순 (대소문자 구별 X), 오름차순 정렬
lists_sort([], "IGNORE_CASE", True) # 알파벳순 (대소문자 구별 X), 내림차순 정렬
```

아래 예제에서는 [gamma, beta, alpha] 리스트를 **알파벳 오름차순**으로 정렬하여 sortLetters 리스트를 생성하는 예제입니다.



출력 결과는 [alpha, beta, gamma] 입니다.

Javascript 코드

```javascript
var letters, sortLetters;

function listsGetSortCompare(type, direction) { // 문자 비교 함수
    var compareFuncs = {
        'NUMERIC': function(a, b) {
                return Number(a) - Number(b); },
        'TEXT': function(a, b) {
                return String(a) > String(b) ? 1 : -1; },
        'IGNORE_CASE': function(a, b) {
                return String(a).toLowerCase() > String(b).toLowerCase() ? 1 : -1; },
    };
    var compare = compareFuncs[type];
    return function(a, b) { return compare(a, b) * direction; };
}

letters = ['gamma', 'beta', 'alpha'];
sortLetters = letters.slice().sort(listsGetSortCompare("TEXT", 1)); // 알파벳 순 오름차순 정렬
window.alert(sortLetters); // ['alpha','beta','gamma']
```

Python 코드

```python
letters = None
sortLetters = None

def lists_sort(my_list, type, reverse): # 문자열 정렬 함수
    def try_float(s):
        try:
            return float(s)
        except:
            return 0
    key_funcs = {
        "NUMERIC": try_float,
```

```
        "TEXT": str,
        "IGNORE_CASE": lambda s: str(s).lower()
    }
    key_func = key_funcs[type]
    list_cpy = list(my_list)
    return sorted(list_cpy, key=key_func, reverse=reverse)

    letters = ['gamma', 'beta', 'alpha']
    sortLetters = lists_sort(letters, "TEXT", False) # 알파벳 순 오름차순 정렬
    print(sortLetters) # ['alpha','beta','gamma']
```

뒤집기

리스트의 요소 순서를 **역순으로 변경**하는 블록입니다.

Javascript 코드

```
[].slice().reverse(); // 리스트 역순으로 변경
```

Python 코드

```
list(reversed([])) # 리스트 역순으로 변경
```

아래 예제에서는 정렬된 sortLetters 리스트를 뒤집어 [gamma, beta, alpha] 리스트를 생성합니다.

Javascript 코드

```
var letters, sortLetters;

function listsGetSortCompare(type, direction) { // 문자 비교 함수
```

```
    var compareFuncs = {
        'NUMERIC': function(a, b) {
                return Number(a) - Number(b); },
        'TEXT': function(a, b) {
                return String(a) > String(b) ? 1 : -1; },
        'IGNORE_CASE': function(a, b) {
                return String(a).toLowerCase() > String(b).toLowerCase() ? 1 : -1; },
    };
    var compare = compareFuncs[type];
    return function(a, b) { return compare(a, b) * direction; };
}

letters = ['gamma', 'beta', 'alpha'];
sortLetters = letters.slice().sort(listsGetSortCompare("TEXT", 1)); // 알파벳 기준 오름차순 정렬
['alpha','beta','gamma']
sortLetters.slice().reverse(); // 리스트 역순으로 변경 ['gamma','beta','alpha']
```

Python 코드

```
letters = None
sortLetters = None

def lists_sort(my_list, type, reverse): # 문자열 정렬 함수
    def try_float(s):
        try:
            return float(s)
        except:
            return 0
    key_funcs = {
        "NUMERIC": try_float,
        "TEXT": str,
        "IGNORE_CASE": lambda s: str(s).lower()
    }
    key_func = key_funcs[type]
```

```
    list_cpy = list(my_list)
    return sorted(list_cpy, key=key_func, reverse=reverse)

letters = ['gamma', 'beta', 'alpha']
sortLetters = lists_sort(letters, "TEXT", False) # 알파벳 기준 오름차순 정렬
['alpha','beta','gamma']
list(reversed(sortLetters)) # 리스트 역순으로 변경 ['gamma','beta','alpha']
```

색상

색상은 다양한 그래픽 프로그램에서 사용됩니다.

블록

팔레트에서 색상 선택

가장 간단한 색상을 얻는 방법은 **색상 선택기**를 사용하는 것입니다. 이 블록은 흰색의 둥근 사각형으로 표시됩니다. 클릭하면 색상 팔레트가 나타나며, 원하는 색상을 선택할 수 있습니다.

Javascript 코드

```
[255, 0, 0]; // 기본 색상 검은색
[255, 0, 0]; // 기본 색상 빨간색
[255, 255, 0]; // 기본 색상 노란색
[0, 255, 0]; // 기본 색상 초록색
[0, 255, 255]; // 기본 색상 청록색
[0, 0, 255]; // 기본 색상 파란색
```

```
[255, 0, 255]; // 기본 색상 분홍색
[255, 255, 255]; // 기본 색상 하얀색
```

Python 코드

```
[255, 0, 0] # 기본 색상 검은색
[255, 0, 0] # 기본 색상 빨간색
[255, 255, 0] # 기본 색상 노란색
[0, 255, 0] # 기본 색상 초록색
[0, 255, 255] # 기본 색상 청록색
[0, 0, 255] # 기본 색상 파란색
[255, 0, 255] # 기본 색상 분홍색
[255, 255, 255] # 기본 색상 하얀색
```

빨강, 초록, 파랑 (RGB) 값으로 색상 만들기

색상 설정 블록을 사용하면 원하는 빨강, 초록, 파랑의 비율을 지정할 수 있습니다.
아래 예제에서는 빨강과 파랑을 최대로 설정하고 초록을 0 으로 설정하여 보라색을 만듭니다.

각 색상 요소의 범위는 0 에서 255(포함) 까지입니다.

Javascript 코드

```
[255, 0, 255]; // R : 255, G : 0, B : 255
```

Python 코드

```
[255, 0, 255] # R : 255, G : 0, B : 255
```

슬라이더를 사용한 색상 생성

이 블록은 **슬라이더**를 이용해 색상을 선택하는 기능을 제공합니다.

사용자는 슬라이더를 조절하여 원하는 **색상**을 직접 조합할 수 있습니다.

각 슬라이더는 빨강 (R), 초록 (G), 파랑 (B) 값 조정을 담당하며, 오른쪽 버튼을 이용해 **명도** (밝기) 를 조절할 수 있습니다.

슬라이더 값을 변경하면 즉시 반영되어, 선택한 색상이 R, G, B 영역에 실시간으로 표시됩니다.

Javascript 코드

```
[90, 85, 224]; // R : 90, G : 85, B : 224
```

Python 코드

```
[90, 85, 224] # R : 90, G : 85, B : 224
```

랜덤 색상 생성

랜덤 색상 블록은 호출될 때마다 무작위 색상을 생성합니다.

이 블록은 빨강, 초록, 파랑 값을 각각 0~255(포함) 사이의 랜덤한 값으로 설정합니다.

Javascript 코드

```
__randomColor(); // 무작위 색상 생성
```

Python 코드

```
__randomColor() # 무작위 색상 생성
```

기술적 세부 사항

블록 컴포저에서 색상은“rr,gg,bb”형식의 텍스트로 표현됩니다.
여기서“rr”, “gg”, “bb”는 각각 빨강, 초록, 파랑 값 (0 ~ 255) 을 나타냅니다.
일반적으로 이 형식은 사용자가 직접 볼 일이 없지만, 아래 프로그램을 실행하면 확인할 수 있습니다.

위 프로그램은 "255,255,255" 을 출력합니다.

Javascript 코드

```
window.alert([255, 255, 255]); // "255,255,255" 출력
```

Python 코드

```
print([255, 255, 255]) # "255,255,255" 출력
```

여담으로, 빛을 혼합하는 방식은 페인트를 혼합하는 방식과 다릅니다.
빨강, 초록, 파랑 빛을 동일한 비율로 혼합하면 흰색이 되지만, 페인트를 섞으면 탁한 색이 나옵니다. —

소리

소리 블록을 활용하면 다양한 효과음과 음성을 재생할 수 있습니다.

블록

소리 재생하기

소리 재생하기 블록은 원하는 소리를 지정한 **볼륨**으로 재생하는 기능을 수행합니다. - **볼륨 조절:** 소리 크기를 조절할 수 있습니다. - **반복 재생:** 반복 체크박스를 활성화하면 선택한 소리를 계속해서 반복 재생할 수 있습니다.

Javascript 코드

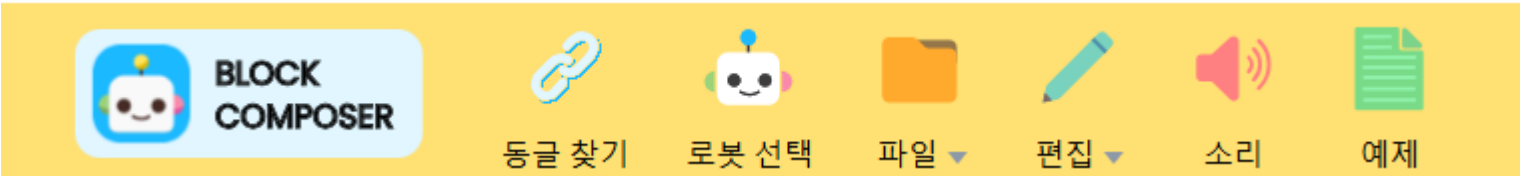
```javascript
__playSound('', 100, false);
```

Python 코드

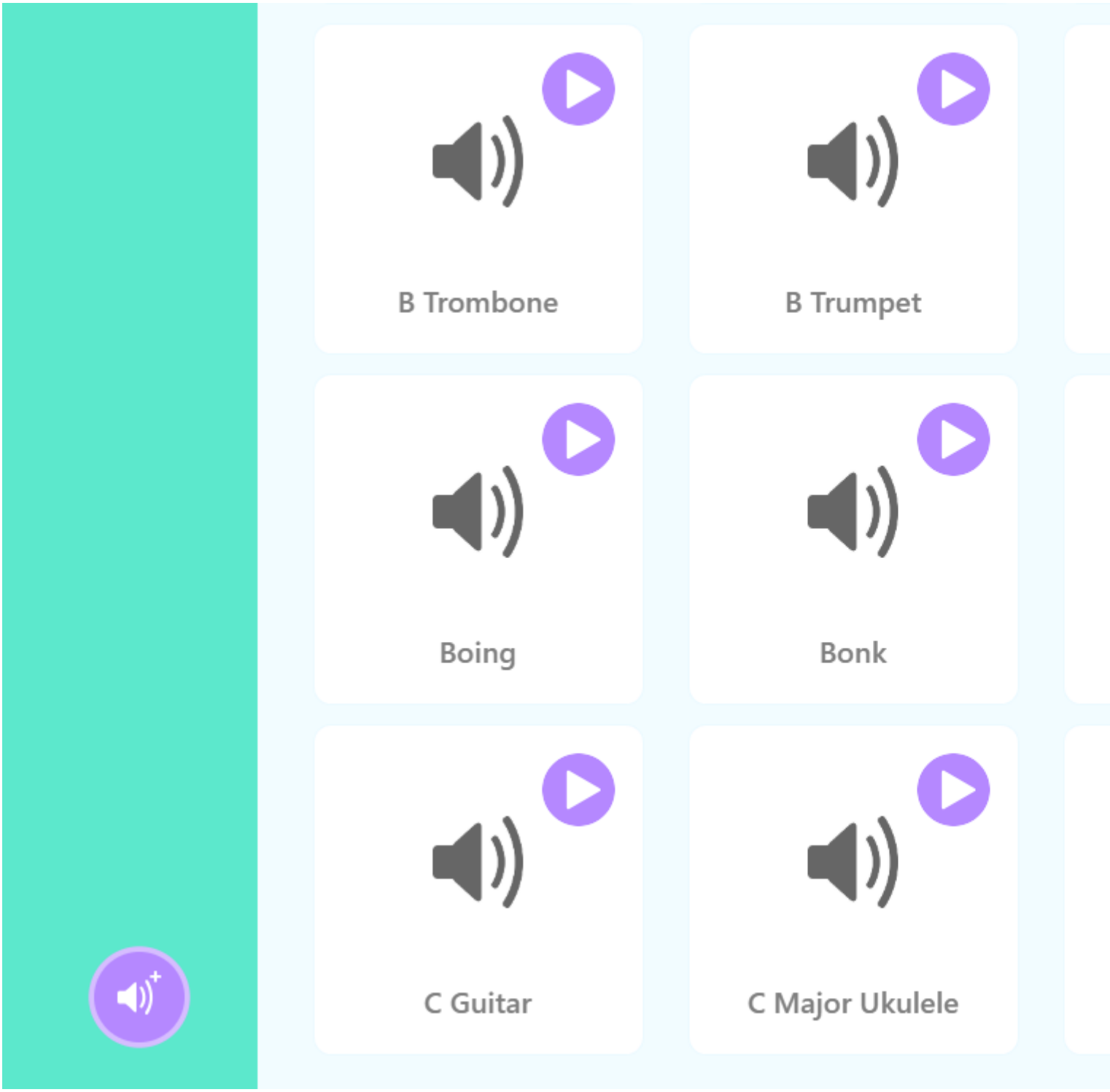
```python
__playSound('', 100, False)
```

소리 추가

상단의 **소리 메뉴**에서 원하는 소리를 선택할 수 있습니다.



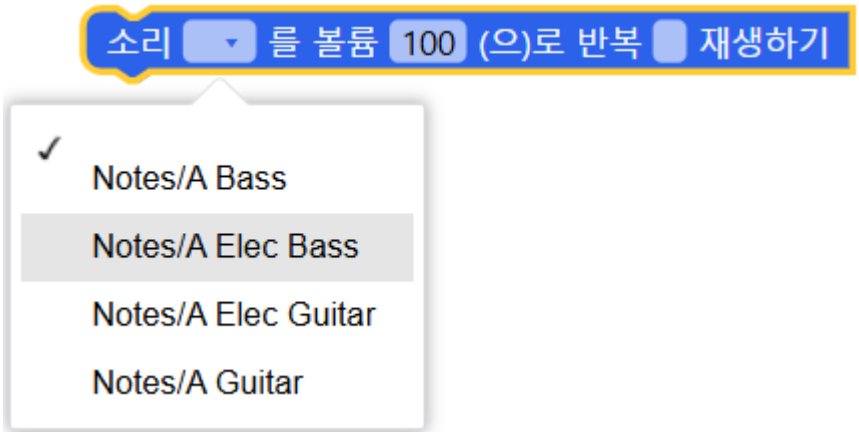
목록에서 소리를 선택하기 전에 먼저 확인하고 직접 들어볼 수 있습니다.



소리 추가 기능을 활용하면 프로젝트에 원하는 효과음을 직접 추가할 수 있습니다. - 추가된 소리는 왼쪽 목록에서 확인할 수 있습니다. - 목록에서 선택한 소리를 재생하여 미리 들어볼 수 있으며, 필요 없는 소리는 삭제할 수도 있습니다.

소리 선택

블록 아래 화살표를 클릭하면 추가한 소리들을 확인하고 선택하여 사용할 수 있습니다.



Javascript 코드

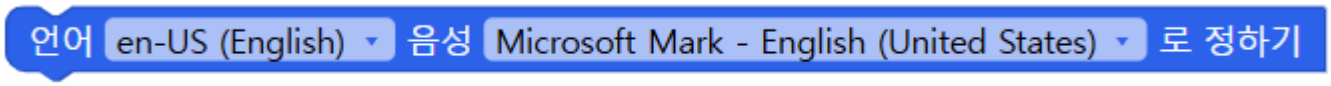
```
__playSound('Notes/A Elec Bass', 100, false); // Notes/A Elec Bass 소리 선택
```

Python 코드

```
__playSound('Notes/A Elec Bass', 100, False) # Notes/A Elec Bass 소리 선택
```

언어 음성 정하기

소리를 재생할 **언어**와 **음성**을 설정하는 블록입니다.
다양한 언어 및 목소리를 선택하여 더욱 자연스러운 음성을 출력할 수 있습니다.



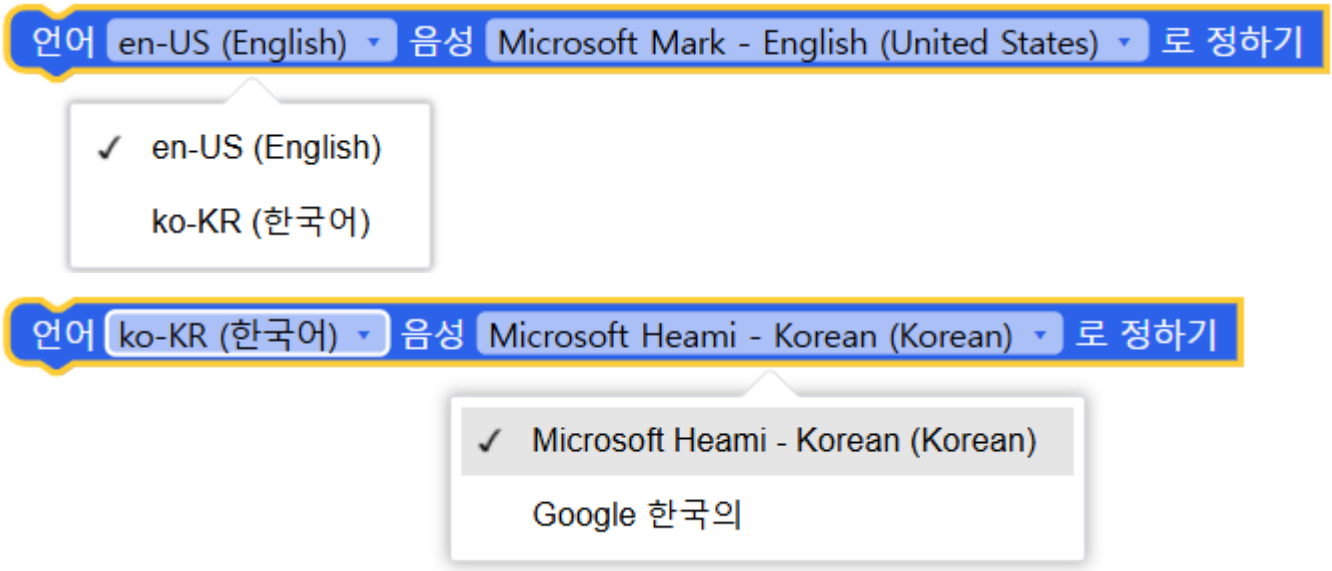
Javascript 코드

```
__setTTSOption('en-US', 'Microsoft Mark - English (United States)'); // 영어, Microsoft Mark 음성 사용하기
```

Python 코드

```
__setTTSOption('en-US', 'Microsoft Mark - English (United States)') # 영어, Microsoft Mark 음성 사용하기
```

- 언어 선택: 원하는 언어를 선택할 수 있습니다.
- 목소리 선택: 다양한 목소리 중에서 원하는 음성을 선택할 수 있습니다.



Javascript 코드

```
__setTTSOption('ko-KR', 'Microsoft Heami - Korean (Korean)'); // 한국어, Heami 음성 사용하기
```

Python 코드

```
__setTTSOption('ko-KR', 'Microsoft Heami - Korean (Korean)') # 한국어, Heami 음성 사용하기
```

다음을 말하기

입력된 텍스트를 소리로 변환하여 말하는 블록입니다.

이 블록을 활용하면 프로젝트에서 원하는 **문장을 음성으로 출력**할 수 있습니다.

- 텍스트 입력: 원하는 문장을 입력하면 해당 문장을 음성으로 변환하여 재생합니다.

Javascript 코드

```
__speak(''); // '' 문장 음성 출력
```

Python 코드

```
__speak('') # '' 문장 음성 출력
```

제어

블록 코딩에서 **제어 블록**은 프로그램의 흐름을 조작하는 역할을 합니다.
일정 시간 대기하거나, 키보드 입력 감지, 로그 출력 등의 기능을 수행할 수 있습니다.

블록

기다리기

기다리기 블록은 이전 명령을 수행한 후 **일정 시간 동안 대기**한 후 다음 명령을 실행하는 기능을 합니다.
이 블록을 사용하면 특정 동작을 일정한 간격으로 실행하거나, 시간 차이를 두고 실행할 수 있습니다.

이 블록이 실행되면, **x** 초 동안 멈춘 후 다음 명령을 실행합니다.

Javascript 코드

```
await __wait(x * 1000);
```

Python 코드

```
import asyncio

await asyncio.sleep(x)
```

1 프레임 기다리기

1 프레임 기다리기 블록은 프로그램의 실행을 한 프레임 (약 0.001 초) 동안 멈춘 후 다음 명령을 실행합니다.
프레임 단위로 프로그램을 제어할 때 유용합니다.

이 블록이 실행되면, 한 프레임 (약 0.001 초) 동안 멈춘 후 다음 명령을 실행합니다.

Javascript 코드

```
await __wait(1);
```

Python 코드

```
import asyncio

await asyncio.sleep(0.001)
```

키 다운 / 키 업

- **키 다운**: 특정 **키를 눌렀을 때** 동작을 수행
- **키 업**: 특정 **키에서 손을 뗄 때** 동작을 수행

키 다운, **키 업** 코드는 아래와 같습니다.

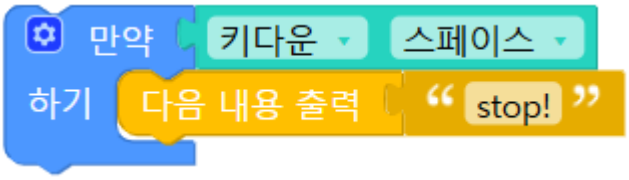
Javascript 코드

```
__keydown(32); // 스페이스바 키 다운
__keyup(32); // 스페이스바 키 업
```

Python 코드

```
__keydown(32) # 스페이스 바 키 다운
__keyup(32) # 스페이스바 키 업
```

스페이스바를 눌렀을 때“stop”을 출력하는 블록입니다.



Javascript 코드

```
if (__keydown(32)) {
    window.alert('stop!');
}
```

Python 코드

```
if __keydown(32):
    print('stop!')
```

키 입력

키 입력 블록은 여러 개의 **키를 동시에 눌렀을 때** 동작을 수행하도록 설정할 수 있습니다.
이 블록을 활용하면 shift, ctrl, alt 와 같은 조합 키를 포함하여 최대 5 개까지 감지할 수 있습니다.

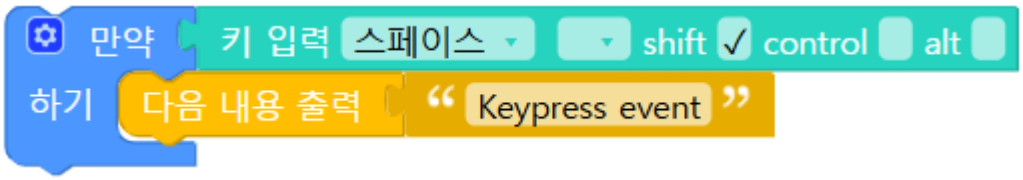
Javascript 코드

```
__keypressed([32, 38, 0, 0, 0]) // 32(스페이스 바), 38(위쪽 방향 키) 를 동시에 눌렀을 때
```

Python 코드

```
__keypressed([32, 38, 0, 0, 0]) # 32(스페이스 바), 38(위쪽 방향 키) 를 동시에 눌렀을 때
```

Shift + Space 키가 눌리면“Keypress event”를 출력하는 블록입니다.



Javascript 코드

```
if (__keypressed([32, 0, 16, 0, 0])) { // Shift + Space 키 입력 시
    window.alert('Keypress event');
}
```

Python 코드

```
if __keypressed([32, 0, 16, 0, 0]): #  Shift + Space 키 입력 시
    print('Keypress event')
```

로그 출력하기

로그 출력하기 블록은 특정 변수나 속성 값을 실시간으로 **콘솔** 창에 출력하여 프로그램 동작을 분석할 수 있도록 합니다. 디버깅 과정에서 값을 확인하거나 데이터 변화를 추적할 때 유용합니다.

Javascript 코드

```
__log(1, 'A', 'K') // 'A' 태그의 K 단위 1 값 로그 출력
```

Python 코드

```
__log(1, 'A', 'K') # 'A' 태그의 K 단위 1 값 로그 출력
```

랜덤한 정수를 콘솔에서 확인하는 예시입니다.

결과는 다음과 같습니다.

로그 스코프 카메라

```
[A] 9K
[A] 100K
[A] 98K
[A] 4K
[A] 86K
[A] 3K
[A] 61K
[A] 23K
[A] 70K
[A] 42K
[A] 56K
[A] 36K
[A] 28K
```

Javascript 코드

```
var random;

function mathRandomInt(a, b) { // 무작위 수 생성 함수
    if (a > b) {
        var c = a;
        a = b;
        b = c;
```

```
    }
    return Math.floor(Math.random() * (b - a + 1) + a);
}

random = mathRandomInt(1, 100);
__log(random, 'A', 'K'); // 1 ~ 100 사이 random 정수 'A' 태그 'K'단위 로그 출력하기
```

Python 코드

```
import random

random2 = None
random2 = random.randint(1, 100)
__log(random2, 'A', 'K') # 1 ~ 100 사이 random 정수 'A' 태그 'K'단위 로그 출력하기
```

스코프 출력하기

스코프 출력하기 블록은 특정 값의 변화를 실시간 그래프 형태로 표현하여 **스코프** 창에서 데이터를 직관적으로 분석할 수 있도록 합니다.
그래프의 색상, 최소/최대 값, 범위를 설정할 수 있으며, 센서 **데이터 시각화**나 변수 변화 추적에 유용합니다.

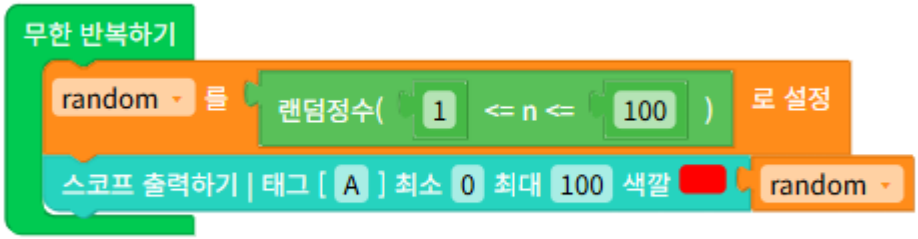
Javascript 코드

```
__scope('_', 0, 1, '#000000', 0); // 0 의 값 0 ~ 1 범위에서 검은 색 스코프 출력
```

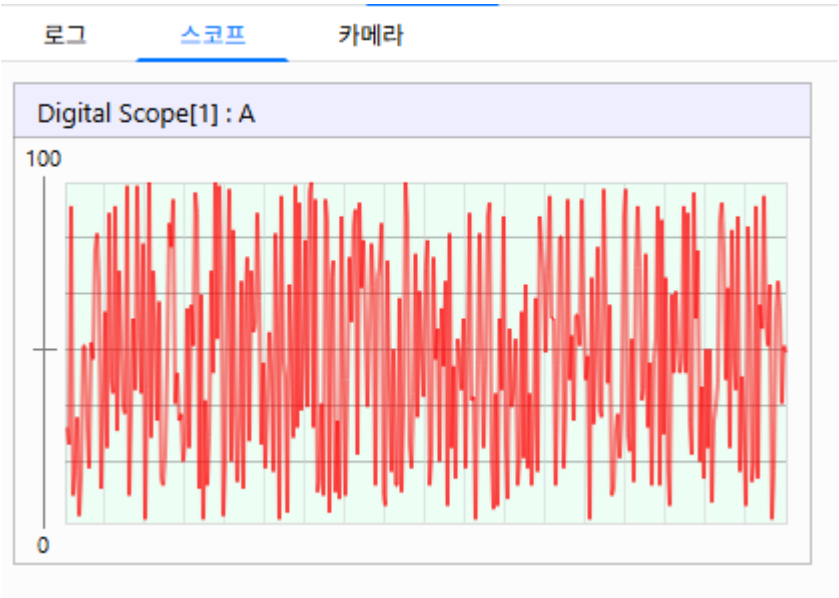
Python 코드

```
__scope('_', 0, 1, '#000000', 0) # 0 의 값 0 ~ 1 범위에서 검은 색 스코프 출력
```

랜덤한 변수 값을 빨간색 그래프로 확인하는 예시입니다.



결과는 다음과 같습니다.



Javascript 코드

```
var random;

function mathRandomInt(a, b) { // 무작위 수 생성 함수
    if (a > b) {
        var c = a;
        a = b;
        b = c;
    }
    return Math.floor(Math.random() * (b - a + 1) + a);
}

random = mathRandomInt(1, 100);
```

```
__scope('A',0, 100, '#ff0000', random); //  0 ~ 100 범위에서 1 ~ 100 사이 random 정수 'A' 태그 빨간색 스코프 출력하기
```

Python 코드

```
import random

random2 = None
random2 = random.randint(1, 100)
__scope('A', 0, 100, '#ff0000', 0) #  0 ~ 100 범위에서 1 ~ 100 사이 random2 정수 'A' 태그 빨간색 스코프 출력하기
```

변수

우리는 변수라는 용어를 수학이나 다른 프로그래밍 언어에서 사용되는 것과 동일한 의미로 사용합니다.
즉, **값을 저장하며 변경할 수 있는 이름을 가진 요소**를 의미합니다.

변수는 여러 가지 방법으로 생성할 수 있습니다.

- 으로 계산 및 각 항목에 대해와 같은 일부 블록은 변수를 사용하며, 해당 변수의 값을 정의합니다.
  이러한 변수는 전통적인 컴퓨터 과학 용어로 **반복 변수 (loop variables)** 라고 합니다.

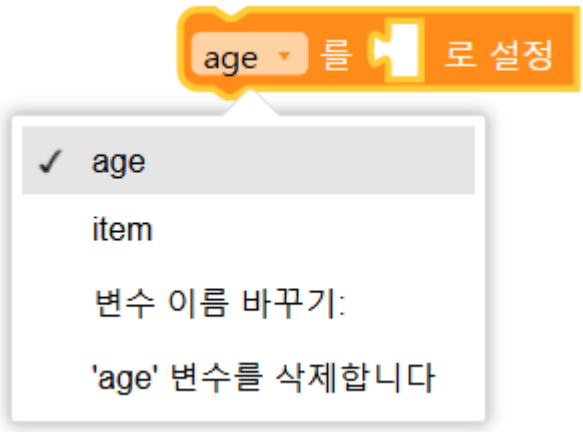
- **사용자 정의 함수** 는 입력값을 정의할 수 있으며, 이는 함수 내에서만 사용할 수 있는 변수 (매개변수 또는 인자) 를 생성합니다.

- 사용자는 **설정** 블록을 통해 언제든지 변수를 생성할 수 있습니다. 이는 전통적으로 "**전역 변수 (global variables)**"라고 불립니다.

- 블록 컴포저는 **지역 변수 (local variables)** 를 지원하지 않습니다.

드롭다운 메뉴

변수의 드롭다운 기호 (삼각형) 를 클릭하면 다음과 같은 메뉴가 나타납니다.



이 메뉴에서 다음 옵션을 선택할 수 있습니다.

- 프로그램에서 정의된 모든 기존 변수 이름이 표시됩니다.
- **"변수 이름 바꾸기"**: 프로그램 전체에서 해당 변수의 이름을 변경합니다. 이 옵션을 선택하면 새 이름을 입력할 수 있는 창이 나타납니다.
- **"변수 삭제"**: 프로그램에서 이 변수를 참조하는 모든 블록을 삭제합니다.

블록

설정

설정 블록은 변수에 값을 할당하며, 해당 변수가 존재하지 않을 경우 새 변수를 생성합니다.
예를 들어, 아래 블록은 "age" 라는 변수를 만들고 값 12 를 할당합니다.

Javascript 코드

```
var age;
age = 12; // age 변수에 12 할당
```

Python 코드

```
age = None
age = 12 # age 변수에 12 할당
```

값 가져오기

아래 블록은 변수에 저장된 값을 제공하지만, 해당 값을 변경하지는 않습니다.

설정 블록 없이 블록을 사용하는 것도 가능하지만, 이는 올바른 프로그래밍 방식이 아닙니다.

Javascript 코드

```
age; // 변수 값 가져오기
```

Python 코드

```
age # 변수 값 가져오기
```

바꾸기

바꾸기 블록은 변수의 값에 숫자를 더하는 역할을 합니다.

위 블록은 다음 코드와 동일한 기능을 수행하는 단축 표현입니다.

Javascript 코드

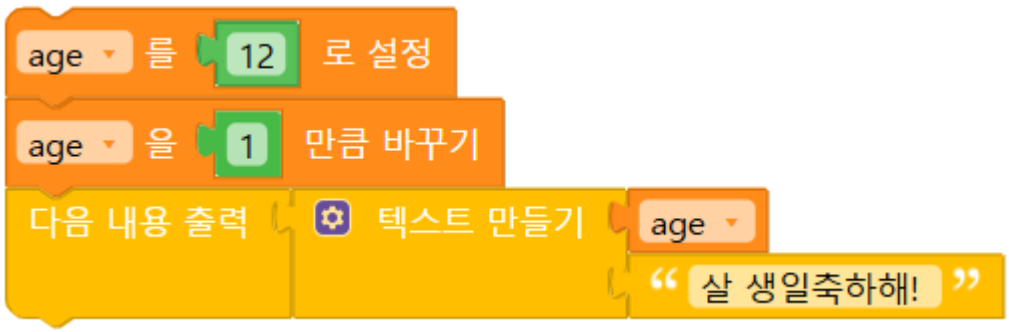
```
var age;
age = (typeof age === 'number' ? age : 0) + 1; // age 1 만큼 더하기
```

Python 코드

```
from numbers import Number


age = None
age = (age if isinstance(age, Number) else 0) + 1 # age 1 만큼 더하기
```

예제

다음은 변수 사용 예제입니다.



첫 번째 줄의 블록은 "age" 라는 변수를 만들고 초기 값을 숫자 12 로 설정합니다.
두 번째 줄의 블록은 값 12 를 가져와서 1 을 더한 후, 그 합 (13) 을 변수에 저장합니다.
마지막 줄은 다음 메시지를 표시합니다: “13 살 생일축하해!”

Javascript 코드

```
var age;


age = 12;
age = (typeof age === 'number' ? age : 0) + 1; // age 1 만큼 더하기
window.alert(String(age) + '살 생일축하해!');
```

Python 코드

```
from numbers import Number


age = None
```

```
age = 12
age = (age if isinstance(age, Number) else 0) + 1 # age 1 만큼 더하기
print(str(age) + '살 생일축하해!')
```

함수

함수 (Function) 는 특정 작업을 수행하는 **블록 (명령어) 들의 모음**입니다.
반복적으로 사용되는 동작을 하나의 함수로 정의하면, 코드를 **간결하고 효율적**으로 관리할 수 있습니다.

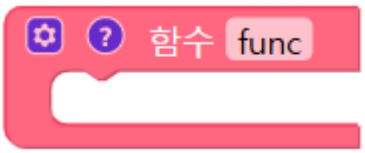
함수는 다음과 같은 특징을 가집니다.
- **재사용 가능**: 한 번 정의하면 여러 번 호출하여 사용할 수 있습니다.
- **입력과 출력**: 매개변수 (입력 값) 를 받아 처리한 후 결과 (출력 값) 를 반환할 수 있습니다.
- **코드의 가독성 향상**: 프로그램의 흐름을 논리적으로 구성할 수 있습니다.

함수 블록

블록 코딩에서는 **함수 블록**을 활용하여 사용자가 직접 원하는 기능을 정의할 수 있습니다.

함수 정의 블록

함수를 정의하는 블록을 사용하면 **새로운 함수**를 만들 수 있습니다.
아래 예제는 func 이라는 이름의 함수를 정의하는 방법을 보여줍니다.



Javascript 코드

```
function func() { // func 함수 정의
}
```

Python 코드

```
def func(): # func 함수 정의
    pass
```

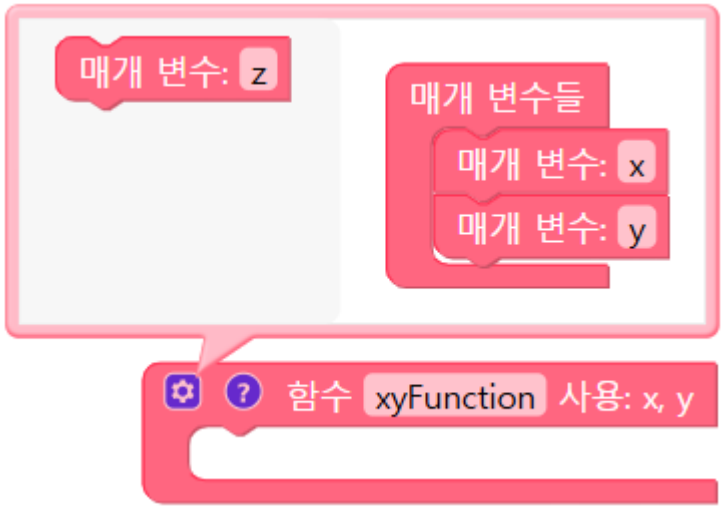
매개변수를 가지는 함수

함수 블록에는 기어 버튼이 있으며, 이를 클릭하여 **매개변수 (입력 값) 를 추가**할 수 있습니다.

- **매개변수 (Parameter)** 는 **함수에 전달하는 입력 값**입니다.

매개변수 설정 방법

1. 매개 변수 블록에서 필요한 변수 (x, y 등) 를 추가합니다.
2. 오른쪽 매개변수 목록에 연결하면, 매개변수를 가지는 함수가 생성됩니다.



매개변수 x, y 를 가지는 함수 `xyFunction` 을 정의하고, 내부에서 매개변수를 활용할 수 있습니다.

Javascript 코드

```javascript
var x, y;

function xyFunction(x, y) { // 매개변수 x, y 를 가지는 xyFunction 정의
}
```

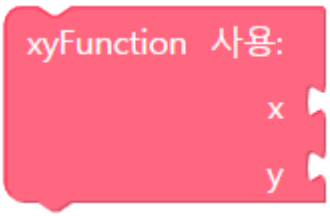
Python 코드

```python
x = None
y = None

def xyFunction(x, y): # 매개변수 x, y 를 가지는 xyFunction 정의
    pass
```

함수 호출 (사용자 정의 함수 블록 생성)

함수를 정의하면, **사용자 정의 함수 블록**이 자동으로 생성됩니다.
이를 통해 **미리 정의한 함수**를 호출하여 실행할 수 있습니다.

다음은 이전에 생성한 xyFunction **함수**를 호출한 블록입니다.



Javascript 코드

```javascript
var x, y;

function xyFunction(x, y) {
}

xyFunction(null, null); // 함수 호출
```

Python 코드

```
x = None
y = None

def xyFunction(x, y):
    pass

xyFunction(None, None) # 함수 호출
```

또한, ? 아이콘을 클릭하면 함수에 대한 **설명 (주석)** 을 추가할 수 있어, 함수를 쉽게 설명할 수 있습니다.

아래 그림은 함수의 **주석**을 추가하는 예시입니다.

Javascript 코드

```
// 이 함수를 설명하세요...
function func() {
}
```

Python 코드

```
# 이 함수를 설명하세요...
def func():
    pass
```

반환값 (Return) 이 없는 함수

반환값이 없는 함수는 특정 동작을 수행하지만 **값을 반환하지 않는 함수**입니다.

다음은“hello”를 출력하는 함수입니다. 이 함수는“hello”를 출력하기만 하고, 특정한 값을 반환하지 않습니다.

Javascript 코드

```javascript
function func() { // 반환값이 없는 함수 정의
    window.alert('hello');
}
```

Python 코드

```python
def func(): # 반환값이 없는 함수 정의
    print('hello')
```

아래 블록을 실행하면, "hello" 가 화면에 출력됩니다.

Javascript 코드

```javascript
function func() {
    window.alert('hello');
}
func(); // 반환값이 없는 함수 호출
```

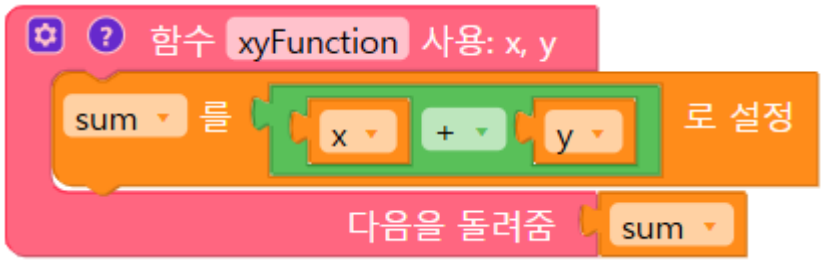
Python 코드

```python
def func():
    print('hello') # 반환값이 없는 함수 호출
func()
```

반환값 (Return) 이 있는 함수

반환값이 있는 함수는 특정 작업을 수행한 후, 결과 **값을 반환**하여 다른 블록에서 활용할 수 있습니다.

아래 함수는 두 숫자의 합을 반환하는 함수입니다. **매개변수 x, y 를 입력받아 x + y 의 결과를 반환**합니다.



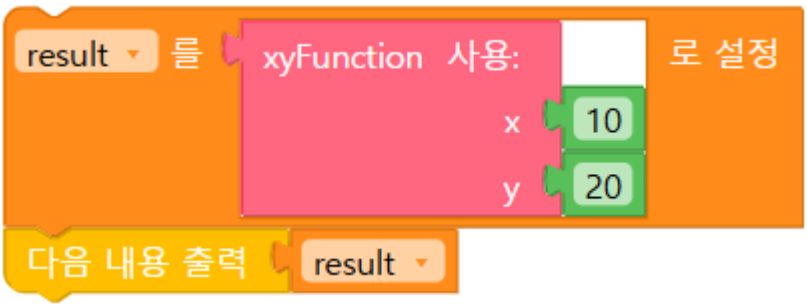
Javascript 코드

```javascript
var x, y, sum;

function xyFunction(x, y) { // 반환값이 있는 함수 정의
    sum = x + y;
    return sum;
}
```

Python 코드

```python
x = None
y = None
sum2 = None # sum 은 예약어 이기 때문에 sum2 사용

def xyFunction(x, y): # 반환값이 있는 함수 정의
    global sum2
    sum2 = x + y
    return sum2
```

아래 함수를 호출하여 결과를 result 변수에 저장하고 사용할 수 있습니다.



함수 실행 예시 (10 + 20 = 30 반환)

30 이 출력됩니다.

Javascript 코드

```javascript
var x, y, result, sum;

function xyFunction(x, y) {
    sum = x + y;
    return sum;
}

result = xyFunction(10, 20); // 반환값이 있는 함수 호출
window.alert(result);
```

Python 코드

```python
x = None
y = None
result = None
sum2 = None # sum 은 예약어 이기 때문에 sum2 사용

def xyFunction(x, y): # 반환값이 있는 함수 호출
    global result, sum2
    sum2 = x + y
    return sum2

result = xyFunction(10, 20)
```

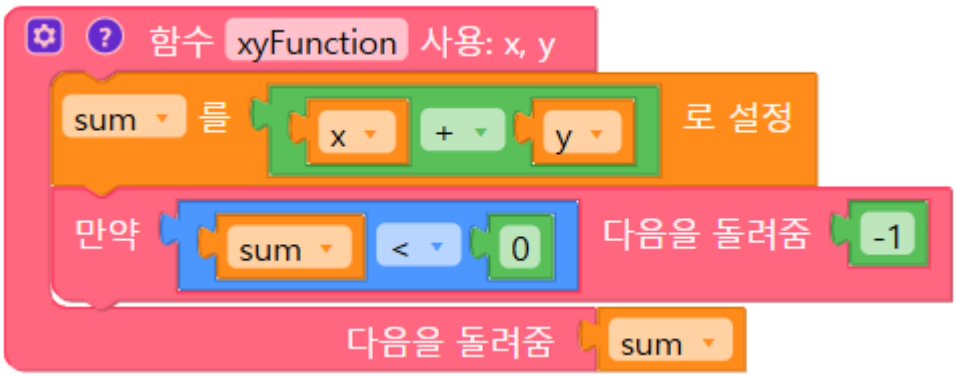
```
print(result)
```

특정 조건에서 값을 반환하는 블록 (만약 다음을 돌려줌 블록)

함수 내에서 **특정 조건을 만족하면 즉시 값을 반환**하고 함수를 종료하는 기능을 수행하는 블록입니다.

이 블록은 **함수 내부에서만 사용**할 수 있으며, 다른 곳에서는 비활성화됩니다.

x 와 y 의 합을 반환하지만, 합이 0 보다 작으면 -1 을 반환하는 함수입니다.



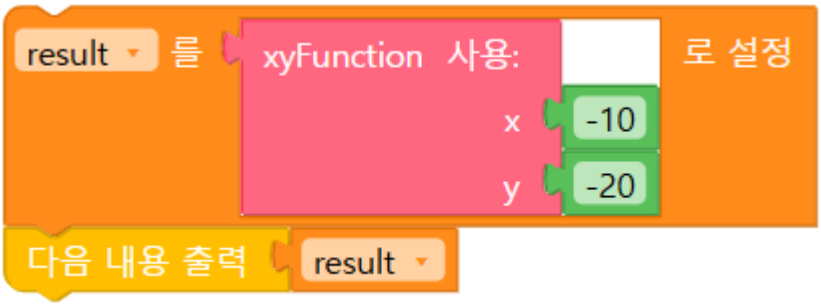
Javascript 코드

```
var x, y, sum;

function xyFunction(x, y) {
    sum = x + y;
    if (sum < 0) { // sum < 0 일 시, -1 반환
        return -1;
    }
    return sum;
}
```

Python 코드

```
x = None
y = None
```

```
sum2 = None # sum 은 예약어 이기 때문에 sum2 사용

def xyFunction(x, y):
    global sum2
    sum2 = x + y
    if sum2 < 0: # sum < 0 일 시, -1 반환
        return -1
    return sum2
```



-1 이 출력됩니다.

Javascript 코드

```
var x, y, result, sum;

function xyFunction(x, y) {
    sum = x + y;
    if (sum < 0) {
        return -1;
    }
    return sum;
}

result = xyFunction(-10, -20); // (-10) + (-20) < 0 이므로 -1 반환
window.alert(result);
```

Python 코드

```
x = None
y = None
result = None
sum2 = None # sum 은 예약어 이기 때문에 sum2 사용

def xyFunction(x, y):
    global result, sum2
    sum2 = x + y
    if sum2 < 0:
        return -1
    return sum2

result = xyFunction(-10, -20) # (-10) + (-20) < 0 이므로 -1 반환
print(result)
```

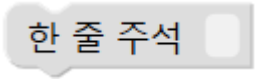
기타

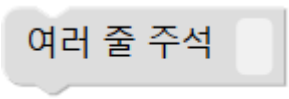
블록 코딩에서 **기타 블록**은 코드 실행 및 로봇의 동작에 영향을 주지 않는 코드들로 구성됩니다.
주석을 달거나, 코드 실행을 종료하는 기능을 수행할 수 있습니다.

블록

주석

주석 블록을 사용하면 코드의 실행에는 영향을 주지 않으면서 **설명**을 추가할 수 있습니다.
주석을 활용하면 코드 가독성이 높아지고, 유지보수가 쉬워집니다.





Javascript 코드

```
// 한 줄 주석

// 여러 줄 주석
// 이어서 입력
```

Python 코드

```
# 한 줄 주석

# 여러 줄 주석
# 이어서 입력
```

링크

링크 블록을 사용하면, **주석** 기능을 활용해 열고 싶은 **페이지 링크**를 추가할 수 있습니다.
열기 버튼을 클릭하면, 입력한 링크의 페이지로 이동할 수 있습니다.

Javascript 코드

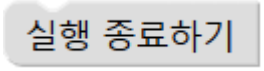
```
// https://google.com
```

Python 코드

```
# https://google.com
```

종료하기

종료하기 블록은 프로그램 실행을 즉시 중단하고 페이지를 새로고침하여 초기 상태로 되돌립니다.
특정 조건에서 강제 종료 기능을 추가할 때 유용합니다.



Javascript 코드

```
__exit(); // 종료하기
```

Python 코드

```
__exit() # 종료하기
```
