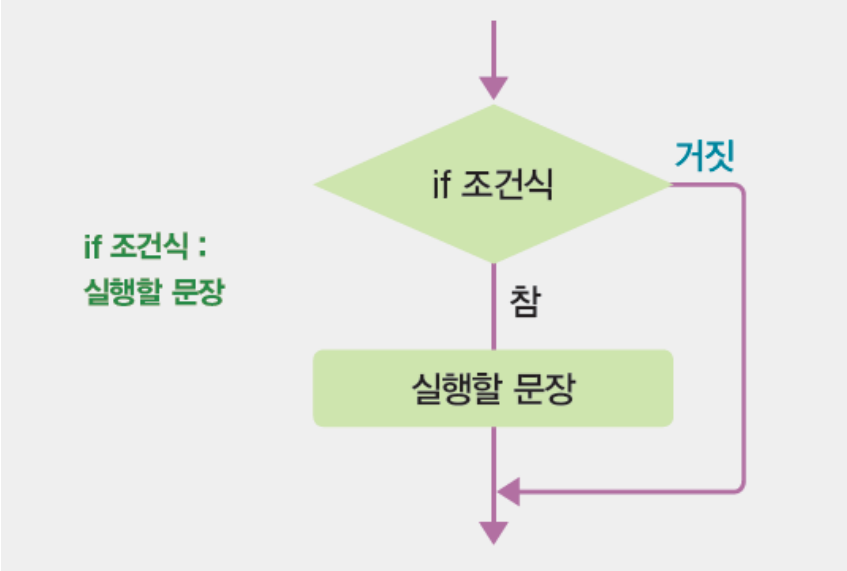
Control Statement

제어문 특징

- 제어문 : 프로그램의 흐름을 제어하는 문장
  - 조건문 : if, elif
  - 반복문 : for, while
  - 분기문 : break, continue

if 문

- if 조건식 : 조건식이 참이면 실행할 문장이 처리되고, 거짓이면 아무것도 실행하지 않고 프로그램을 종료



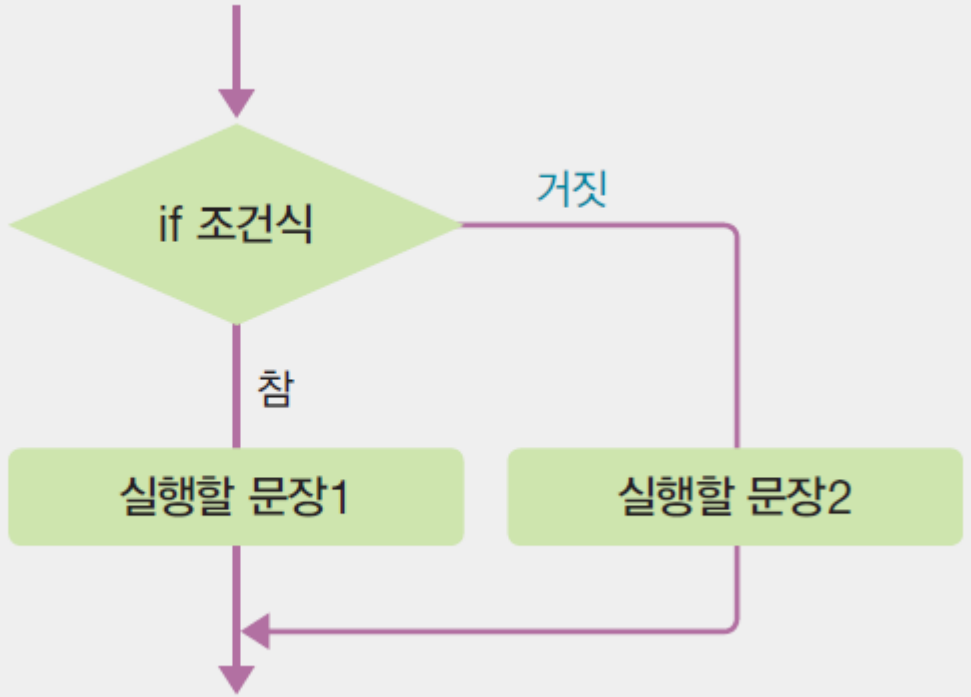

```
a=99
if a<100 :
   print("100보다 작군요.")
```

```
100보다 작군요.
```

if 문

- if~else 문
  - 참일 때 실행하는 문장과 거짓일 때 실행하는 문장이 다를 때 사용

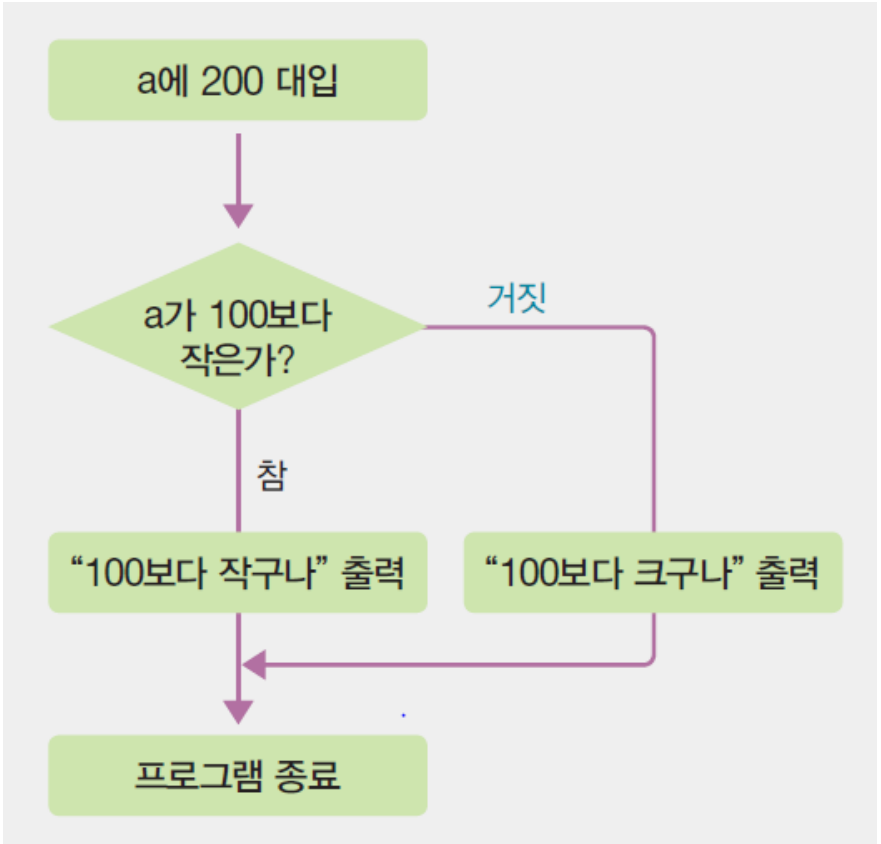
```
if 조건식 :
  실행할 문장1
else :
  실행할 문장2
```



if 문

```
1  a=200
2
3  if a<100 :
4      print("100보다 작군요.")
5  else :
6      print("100보다 크군요.")
```

```
100보다 크군요.
```



if 문

- 입력한 숫자가 짝수인지 홀수인지 계산하는 프로그램

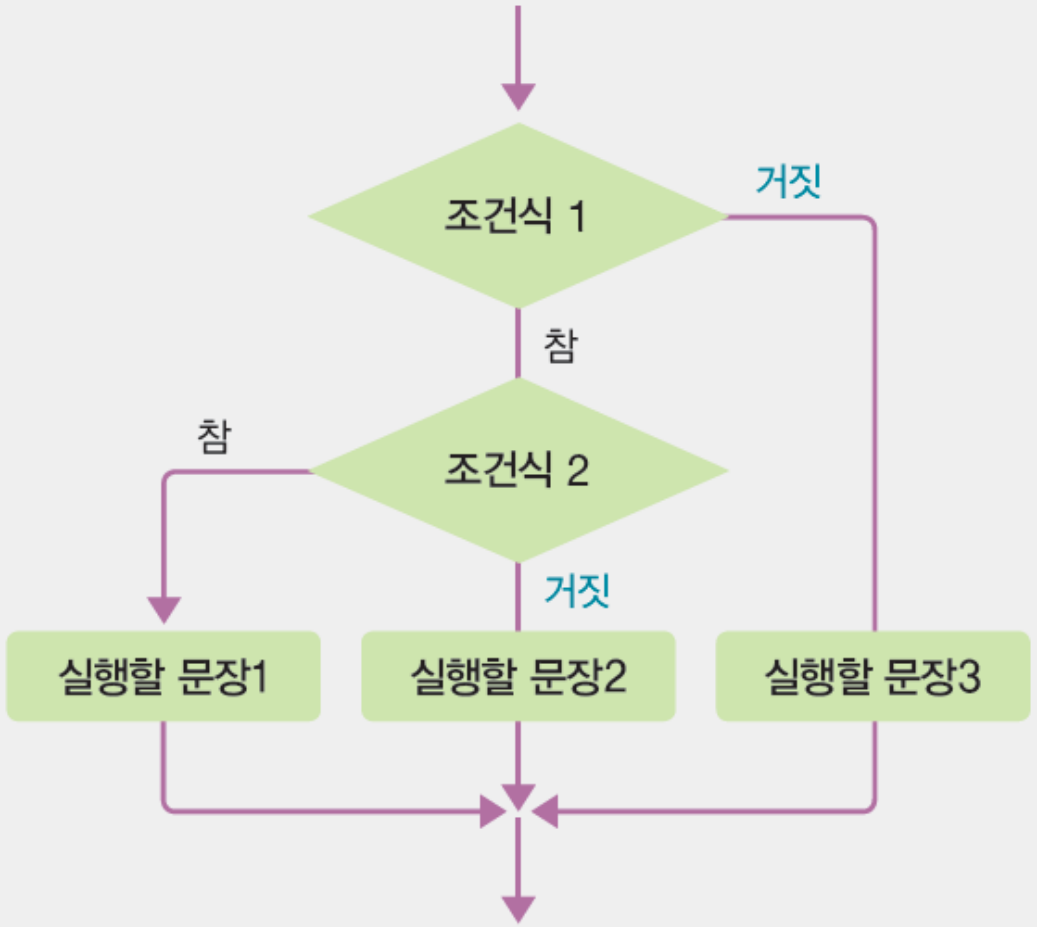
```
1  a=int(input("정수를 입력하세요 : "))
2
3  if a%2==0 :
4      print("짝수를 입력했군요.")
5  else :
6      print("홀수를 입력했군요.")
```

```
정수를 입력하세요 : 125  ← 사용자가 입력한 값
홀수를 입력했군요.
```

중첩 if 문

- if ~ else ~ if ~ else문

```
if 조건식 1 :
    if 조건식 2 :
        실행할 문장1
    else :
        실행할 문장2
else :
    실행할 문장3
```



- 다음과 같이 출력 되도록 간단한 계산기 프로그램 작성

```
첫 번째 숫자를 입력하세요: 5
연산자를 입력하세요 (+, -, *, /): *
두 번째 숫자를 입력하세요: 3
결과: 15.0
```

간단한 계산기 프로그램

```
def calculator():
    num1 = float(input("첫 번째 숫자를 입력하세요: "))
    operator = input("연산자를 입력하세요 (+, -, *, /): ")
    num2 = float(input("두 번째 숫자를 입력하세요: "))

    if operator == '+':
        result = num1 + num2
    elif operator == '-':
        result = num1 - num2
    elif operator == '*':
        result = num1 * num2
    elif operator == '/':
        result = num1 / num2
    else:
        print("올바른 연산자를 입력해주세요.")
        return

    print("결과:", result)


calculator()
```

- 다음과 같이 출력 되도록 종합 계산기 프로그램 작성

```
계산할 수식을 입력하세요: 3 + 5 * 2
수식 계산 결과: 13

시작 숫자를 입력하세요: 1
끝 숫자를 입력하세요: 5
1부터 5까지의 합계: 15
```

- 종합 계산기 프로그램

```
def calculate_expression(expression):
    try:
        result = eval(expression)
        print("수식 계산 결과:", result)
    except:
        print("잘못된 수식입니다.")

def calculate_sum(start, end):
    try:
        start = int(start)
        end = int(end)
        if start > end:
            start, end = end, start  # 시작과 끝 값 교환
        total = sum(range(start, end + 1))
        print(f"{start}부터 {end}까지의 합계:", total)
    except:
        print("잘못된 입력입니다.")

# 수식 계산
expression = input("계산할 수식을 입력하세요: ")
calculate_expression(expression)

# 합계 계산
start_num = input("시작 숫자를 입력하세요: ")
end_num = input("끝 숫자를 입력하세요: ")
calculate_sum(start_num, end_num)
```

- 종합 계산기 프로그램

- calculate_expression 함수는 입력된 수식을 eval() 함수를 사용하여 계산
- eval() 함수는 문자열로 표현된 수식을 파이썬 표현식으로 해석하고 계산 결과를 반환
- try-except 구문을 사용하여 수식이 올바른지 확인하고, 계산 결과를 출력
- 예외가 발생하면 "잘못된 수식입니다."라고 출력

- calculate_sum 함수는 입력된 시작과 끝 값을 정수로 변환하고, 시작과 끝 사이의 모든 수의 합계를 계산
- try-except 구문을 사용하여 입력값이 올바른지 확인하고, sum() 함수와 range() 함수를 사용하여 합계를 계산
- 시작과 끝 값이 역전된 경우에는 변수 교환을 수행하여 올바른 범위로 설정

- 사용자로부터 수식을 입력받아 calculate_expression 함수를 호출하여 계산 결과를 출력

- 사용자로부터 시작 숫자와 끝 숫자를 입력받아 calculate_sum 함수를 호출하여 합계를 계산하고 출력

for 문

- for 문은 반복 작업을 수행할 때 사용되는 제어문
- for 문은 주로 시퀀스형 데이터를 순회하면서 각 요소에 대해 작업을 수행하는 데 사용

```
for 변수 in 시퀀스:
    # 반복할 코드
```

- for 문은 다음과 같은 단계로 동작
  - 시퀀스에 있는 각 요소들을 차례대로 변수에 할당
  - 할당된 변수를 기반으로 반복할 코드 블록이 실행
  - 코드 블록의 실행이 완료되면 다시 시퀀스로 돌아가 다음 요소를 할당하여 반복을 진행

for 문

- for문의 작동
  - range( )함수는 지정된 범위의 값을 반환

```
형식:
for 변수 in range( 시작값, 끝값+1 , 증가값 ) :
    이 부분을 반복
```

  - range(0, 3, 1)은 [0, 1, 2]와 같음

```
for i in range(0, 3, 1) :
    print("안녕하세요? for문을 공부중입니다. ^^")
```

```
for i in [0, 1, 2] :
    print("안녕하세요? for문을 공부중입니다. ^^")
```

```
1회 : i에 0을 대입한 후 print() 수행
2회 : i에 1을 대입한 후 print() 수행
3회 : i에 2를 대입한 후 print() 수행
```

for 문 예제

- 500과 1000 사이에 있는 홀수의 합을 구하는 프로그램

```
1  i, hap=0, 0
2
3  for i in range(501, 1001, 2) :
4      hap=hap+i
5
6  print("500에서 1000까지 홀수의 합 : %d" % hap)
```

```
500에서 1000까지 홀수의 합 : 187500
```

for 문 예제

- 입력한 값까지 for문으로 합계 구하기
  - 사용자가 원하는 값을 입력하여 1부터 입력한 수까지의 합을 구하는 프로그램

```
1  i, hap=0, 0
2  num=0
3
4  num=int(input("값 입력 : "))
5
6  for i in range(1, num+1, 1) :
7      hap=hap+i
8
9  print("1에서 %d까지 합 : %d" % (num, hap))
```

```
값 입력: 100  ← 사용자가 입력한 값
1에서 100까지 합 : 5050
```

중첩 for 문

- 중첩 for문의 개념
  - 중첩 for문은 for문 내부에 또 다른 for문이 들어있는 형태

```
for 변수1 in 시퀀스1:
    # 바깥쪽 반복문 코드

    for 변수2 in 시퀀스2:
        # 안쪽 반복문 코드

    # 바깥쪽 반복문 이어서 실행되는 코드
```

중첩 for 예제

```
for i in range(2, 10):
    print(f"{i}단")
    for j in range(1, 10):
        print(f"{i} x {j} = {i*j}")
    print()
```

```
2단
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18


...

9단
9 x 1 = 9
9 x 2 = 18
9 x 3 = 27
9 x 4 = 36
9 x 5 = 45
9 x 6 = 54
9 x 7 = 63
9 x 8 = 72
9 x 9 = 81
```

중첩 for 예제

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

for row in matrix:
    for element in row:
        print(element, end=" ")
    print()
```

```
1 2 3
4 5 6
7 8 9
```

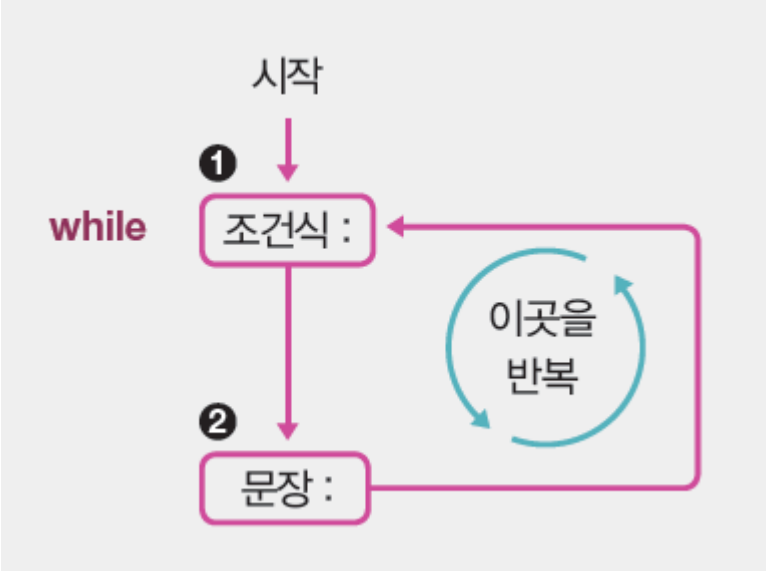
- matrix는 2차원 배열로, 각 행(row)은 대괄호([])로 묶여 있고 쉼표(,)로 구분
- 바깥쪽 for 문에서 matrix의 각 행(row)을 반복
- row는 각 행의 요소를 순차적으로 가리키는 변수
- 안쪽 for 문에서는 현재 행(row)의 요소를 반복하여 출력
- element는 현재 요소를 가리키는 변수
- print(element, end=" ")은 현재 요소(element)를 출력하고 공백으로 끝을 지정
- 안쪽 for 문의 반복이 끝나면, print() 함수를 사용하여 줄 바꿈을 수행
- 바깥쪽 for 문의 반복이 완료되면, 모든 행의 요소 출력이 완료

while 문

- for문과 while문의 비교
  - for문의 형식

```
for 변수 in range(시작값, 끝값+1, 증가값)
```

  - while문은 while문 안의 조건식을 확인하여 값이 참이면 '문장'을 수행. 조건식이 참인 동안 계속 반복함




```
형식:
변수 = 시작값
while 변수값 < 끝값 :
    이 부분을 반복
    변수 = 변수 + 증가값
```

while 문

- While문을 이용한 1에서 10까지의 합

```
1  i, hap=0, 0
2
3  i=1
4  while i<11 :
5      hap=hap+i
6      i=i+1
7
8  print("1에서 10까지의 합 : %d" % hap)
```

```
1에서 10까지의 합 : 55
```

분기문(Jump Statement) break문 사용 예제

- 1~100까지 더하되, 누적 합계(hap)가 1000 이상이 되는 시작 지점을 구하는 프로그램

```
1  hap, i=0, 0
2
3  for i in range(1,101) :
4      hap+=i
5
6      if hap>=1000 :
7          break
8
9  print("1~100의 합에서 최초로 1000이 넘는 위치 : %d" % i)
```

```
1~100의 합에서 최초로 1000이 넘는 위치 : 45
```

분기문(Jump Statement) continue

- **반복문으로 다시 돌아가는 continue문**
  - continue문을 만나면 무조건 블록의 남은 부분을 건너뛰고 반복문의 처음으로 돌아감
  - 1~100까지의 합을 구하되 1 +2 +4 +5 +7 +8 +10 +…과 같이 3의 배수를 건너뛰고(=제외하고)

```
1  hap, i=0, 0
2
3  for i in range(1,101) :
4      if i % 3==0 :
5          continue
6
7      hap+=i
8
9  print("1~100의 합계(3의 배수 제외) : %d" % hap)
```

```
1~100의 합계(3의 배수 제외) : 3367
```