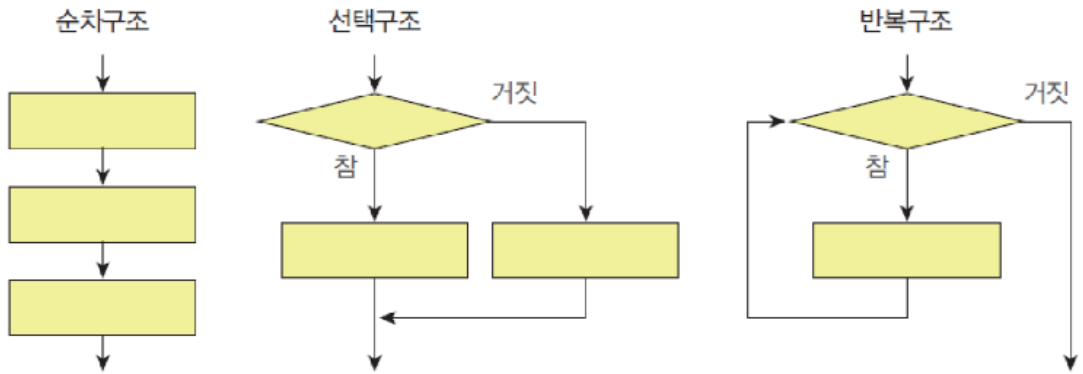
비즈니스데이터분석

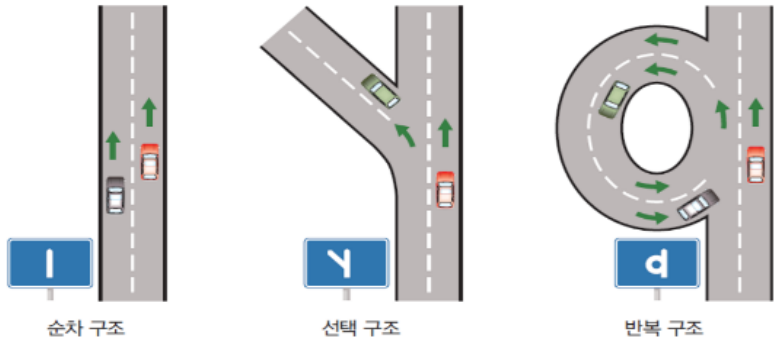
- 파이썬 기초

한양대학교

제어문

순차 구조(sequence) – 명령들이 순차적으로 실행되는 구조이다.
선택 구조(selection) – 둘 중의 하나의 명령을 선택하여 실행되는 구조이다.
반복 구조(iteration) – 동일한 명령이 반복되면서 실행되는 구조이다.





조건문

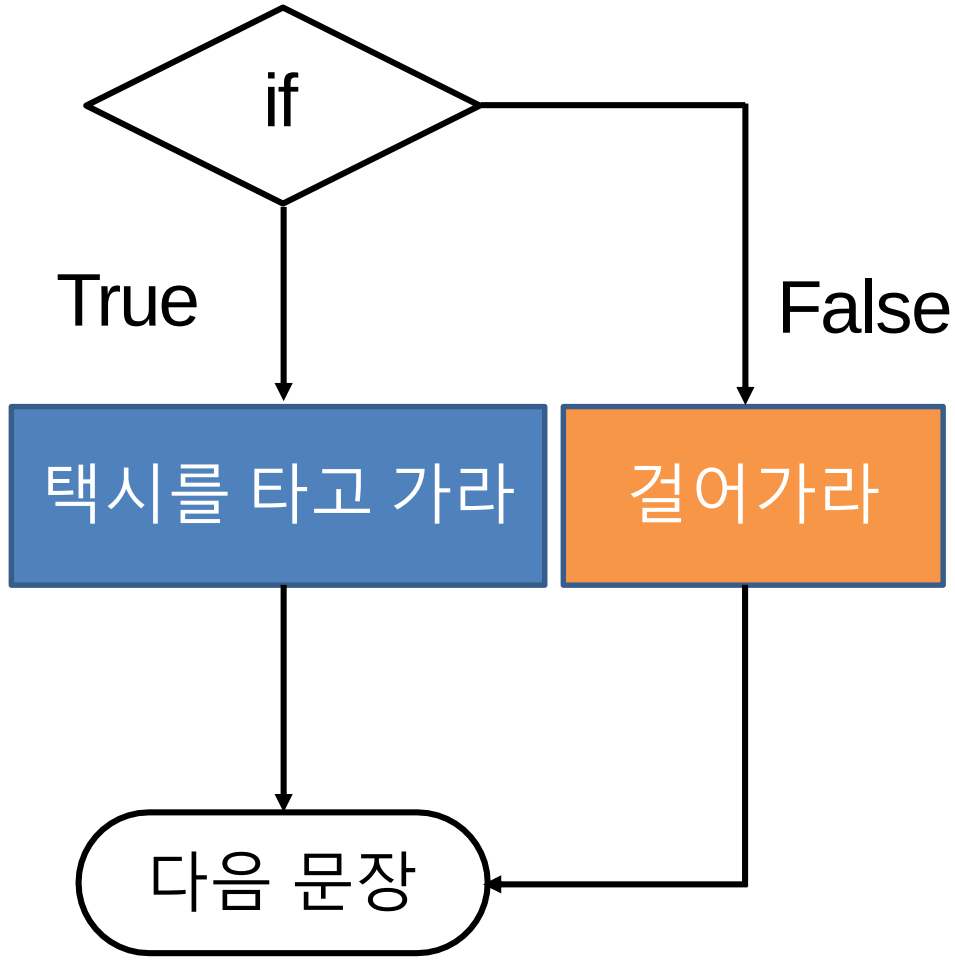
- 특정 조건을 만족하는지 여부에 따라 실행하는 코드가 달라야 할 때 사용
- if – else 문
  - 조건식을 만족하는 경우와 만족하지 않는 경우를 구분하여 코드를 작성할 때 사용

```
if 조건식:
조건식의 결과가 True일 때 실행문
else:
조건식의 결과가 False일 때 실행문
```

if else문

```
money = True

if money:
    print("택시를 타고 가라")
else:
    print("걸어 가라")
```

택시를 타고 가라



if else문

```
money = 2000
pay = 30000

      if false
if money >= pay:
    print("택시를 타고 가라")
else:
    print("걸어 가라")
```

걸어가라

- 비교연산자
  - x < y, x >y, x <= y, x >=y
  - x == y
  - x != y

if else문

```
money = 2000
pay = 30000
card = True

if money >= pay or card:
    print("택시를 타고 가라")
else:
    print("걸어 가라")
```

```
택시를 타고 가라
```

- 비교연산자
  - and, or, not
    - x or y
    - x and y
    - not x

if else문

```
pocket = ["paper", "cellphone", "money"]

if "money" in pocket:
    print("택시를 타고 가라")
else:
    print("걸어 가라")
```

if true

택시를 타고 가라

- 비교연산자
  - in, not in
    - x in 반복가능한 객체
    - x not in 반복가능한 객체

if elif문

- 특정 조건에 따라 3가지 이상으로 구분해야 할 때 사용

```
if 조건식1:
    조건식1의 결과가 True 일 때 실행문
elif 조건식2:
    조건식1의 결과가 False이고, 조건식2의 결과가 True일 때 실행문
elif 조건식3:
    조건식1, 2의 결과가 모두 False이고, 조건식3의 결과가 True일 때 실행문
else:
    조건식1, 2, 3의 결과가 모두 False일 때 실행문
```

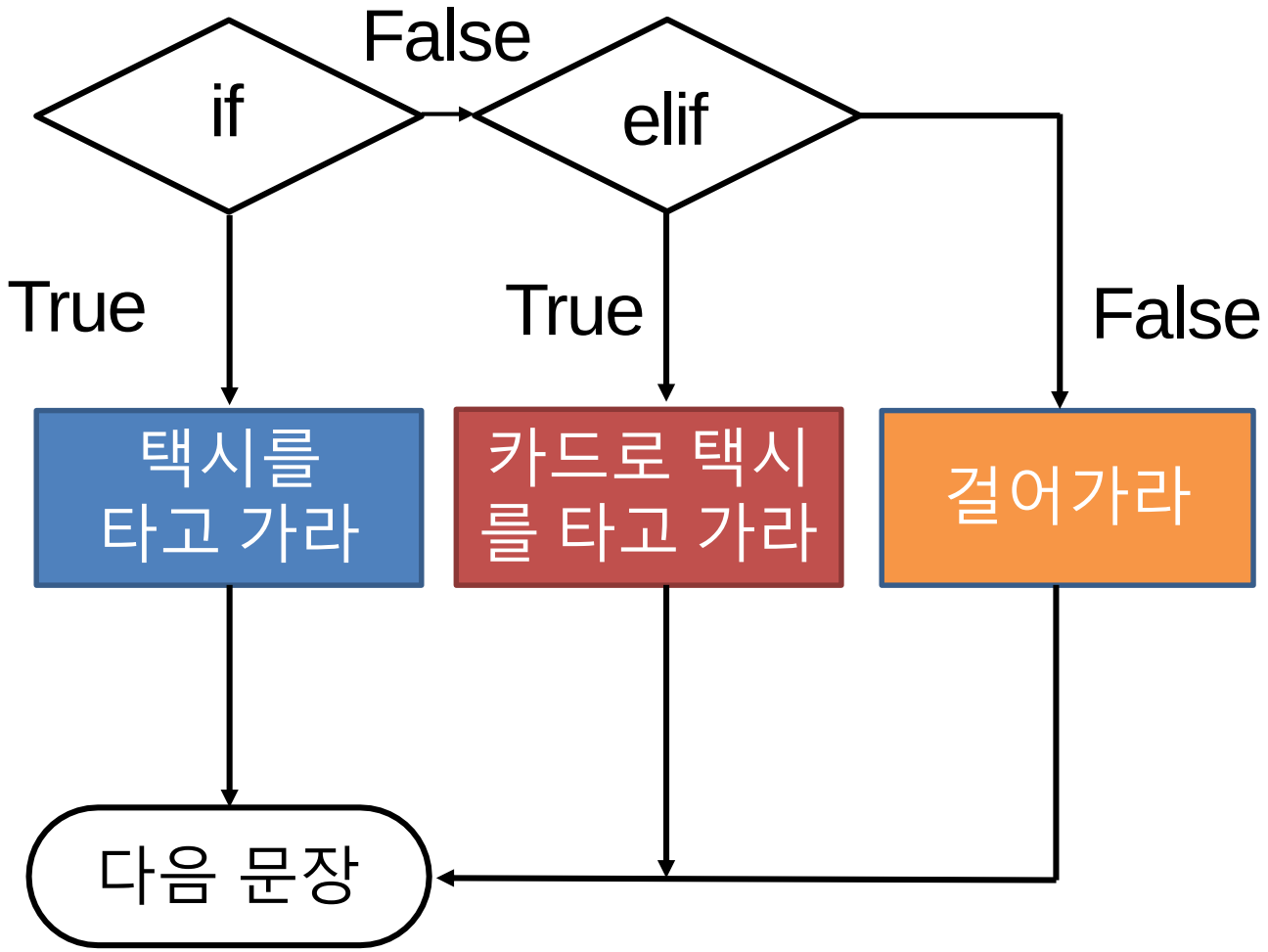
if elif문

```
pocket = ["cellphone"]
card = True
if "money" in pocket:
    print("택시를 타고 가라")
else:
    if card:
        print("택시를 타고 가라")
    else:
        print("걸어 가라")
```

```
# 예) 주머니에 돈이 있으면 택시를 타고
# 주머니에 돈은 없지만 카드가 있다면
    택시를 타고
# 돈도 없고 카드도 없으면 걸어가라
```

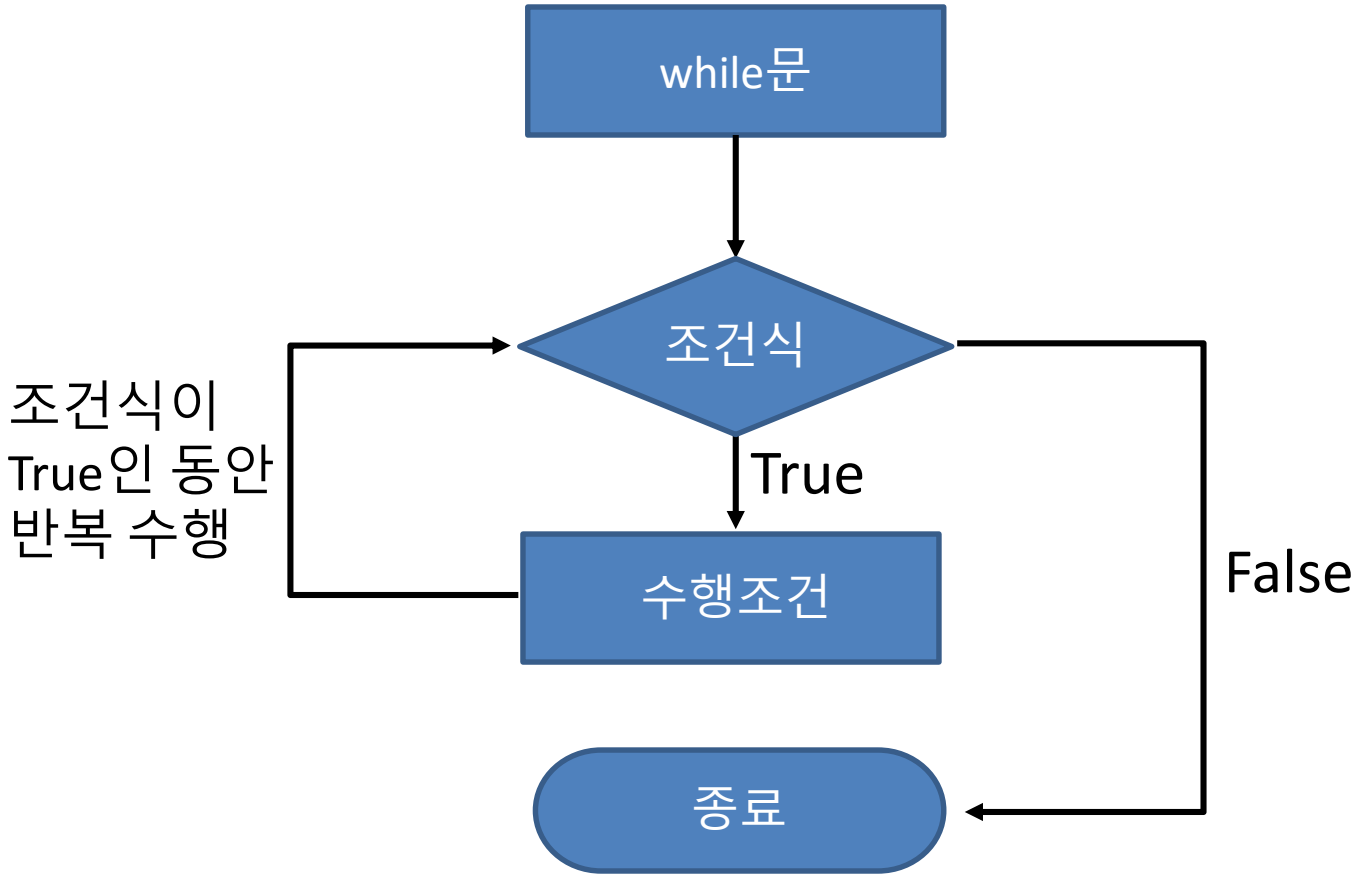
if elif문

```
pocket = ["cellphone"]
card = True

if "money" in pocket:
    print("택시를 타고 가라")
elif card:
    print("카드로 택시를 타고 가라")
else:
    print("걸어 가라")
```



while문

- 값의 범위나 반복 횟수가 정해져 있을 때 주로 사용하는 반복문



while문

- 특정 조건을 만족하는 동안 반복해서 수행해야 할 때
- 조건이 True인 동안 반복해서 실행
- 반복해야 하는 횟수나 값의 범위가 명확하지 않은 경우에 주로 사용

```
while 조건식:
    반복 실행문
```

while문

```
hits = 0
tree_hp = 3

while hits < tree_hp:
hits = 3 hits += 1 <<<< same hits = hits + 1
    print(f"나무를 {hits} 번째 찍었습니다")

    if hits == tree_hp:
        print("나무가 넘어갑니다")
```

hits < tree_hp를 만족하는 동안 조건 판단 후 수행

if hits == tree이면 print문 출력 후 종료

```
나무를 1번째 찍었습니다
나무를 2번째 찍었습니다
나무를 3번째 찍었습니다
나무가 넘어갑니다
```

while문

while 반복문 수행 구조

hits	수행문 hits	수행문	hits	조건식	조건문	수행하는 문장	while 문
0	0 < 3	참	1	1 == 3	거짓	나무를 1번째 찍었습니다	반복
1	1 < 3	참	2	2 == 3	거짓	나무를 2번째 찍었습니다	반복
2	2 < 3	참	3	3 == 3	거짓	나무를 3번째 찍었습니다	반복
				3 == 3	참	나무가 넘어갑니다	종료

while문 중첩

day = 1

day <= 5를
만족하는 동안 수행

day < = 5 인 동안 hour <= 3인 수행문 출력 후 종료

```
while day <= 5:
    hour = 1
    while hour <= 3:
        print(f"{day}일차 {hour}교시 입니다")
        hour += 1
    day += 1
```

1일차 1교시입니다
1일차 2교시입니다
1일차 3교시입니다
2일차 1교시입니다
…
4일차 3교시입니다
5일차 1교시입니다
5일차 2교시입니다
5일차 3교시입니다

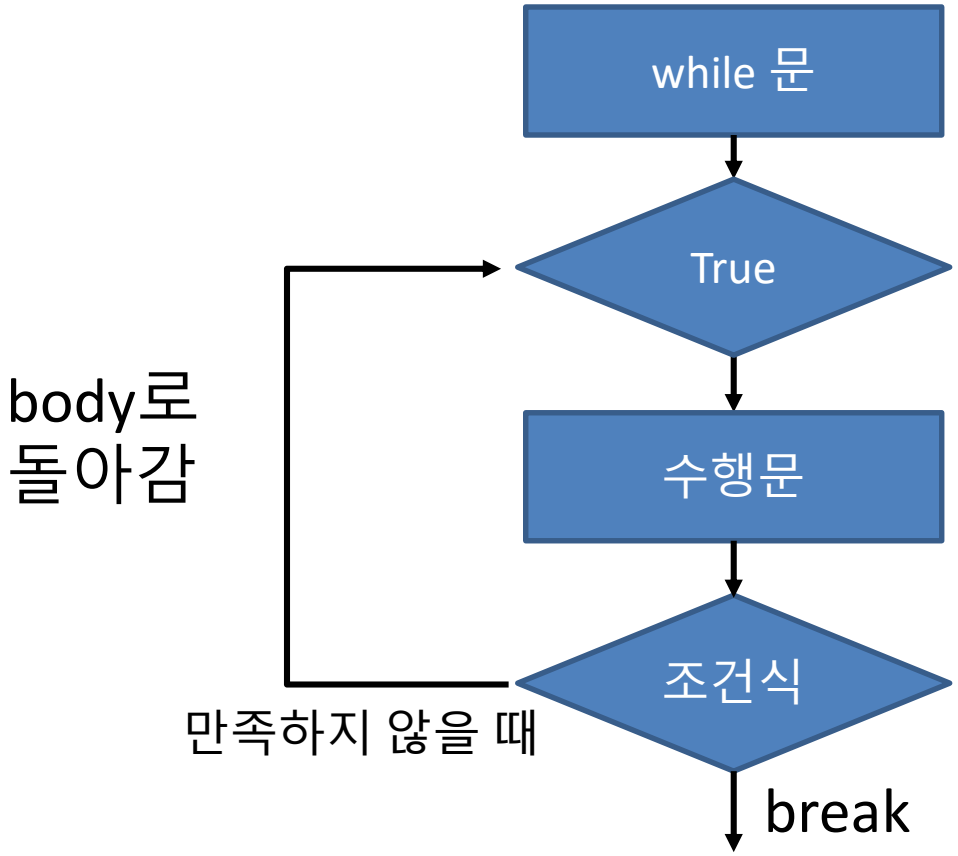
while문

while 반복문 수행 구조

day	수행문	조건판단	hour	수행문	조건판단	수행하는 문장	while 문
1	1 <= 5	참	1	1 <= 3	참	1일차 1교시입니다	hour 반복
1	1 <= 5	참	2	2 <= 3	참	2일차 2교시입니다	hour 반복
1	1 <= 5	참	3	3 <= 3	참	1일차 3교시입니다	hour 반복
1	1 <= 5	참	4	4 <= 3	거짓		day 반복
2	2 <= 5	참	1	1 <= 3	참	2일차 1교시입니다	hour 반복
2	2 <= 5	참	2	2 <= 3	참	2일차 2교시입니다	hour 반복
2	2 <= 5	참	3	3 <= 3	참	2일차 3교시입니다	hour 반복
2	2 <= 5	참	4	4 <= 3	거짓		day 반복
…	…	…		…	…	…	…
5	5 <= 5	참	3	3 <= 3	참	5일차 3교시입니다	hour 반복
5	5 <= 5	참	4	4 <= 3	거짓		day 반복
6	6 <= 5	거짓					종료

while - break문

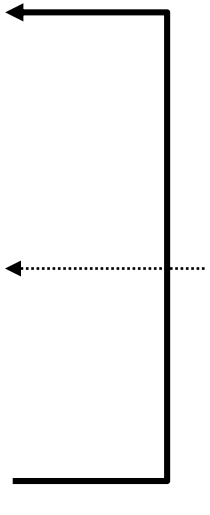
- 값의 범위나 반복 횟수가 정해져 있을 때 반복문에서 강제로 빠져나가는 경우



break문

- 반복문을 강제로 종료시킬 때 사용

```
n = 1

while True:
    print(n)

    if n == 10;
        break

    n += 1
```



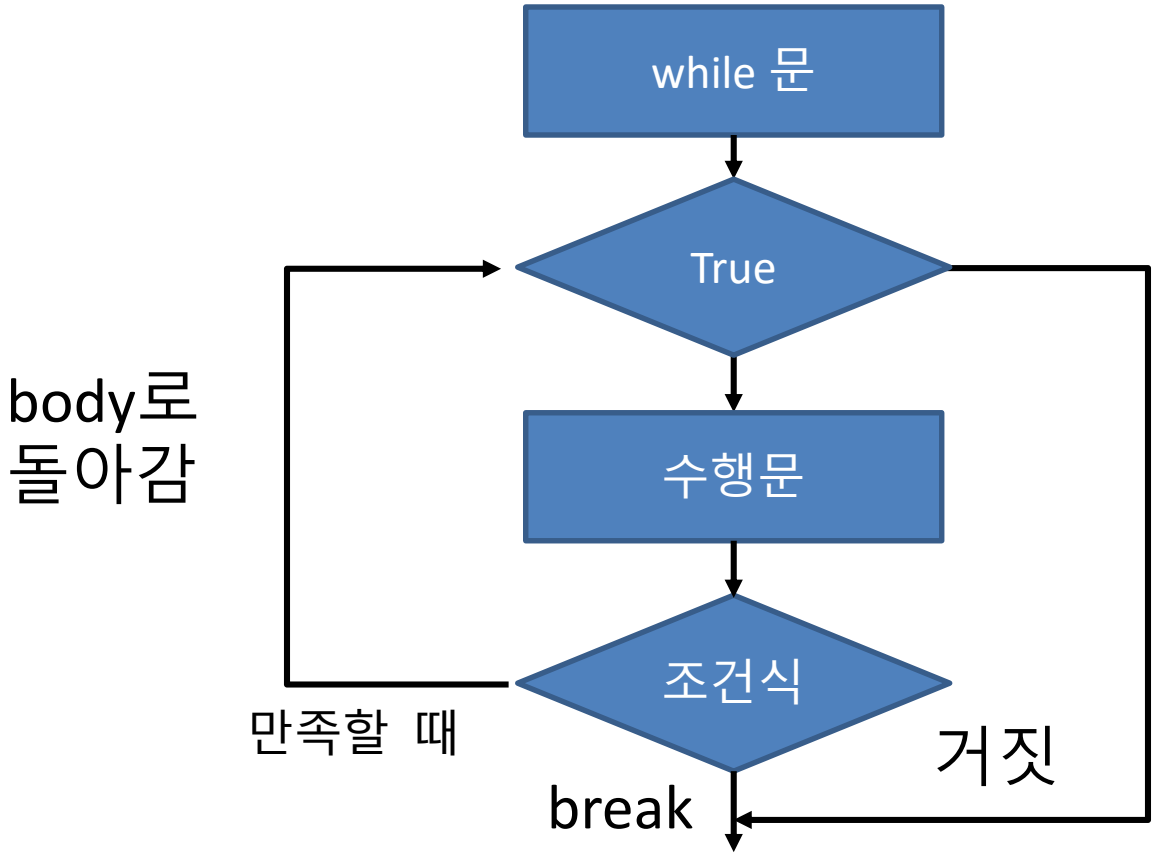
만족하지 않을 때
반복 및 출력

만족하지 않을 때
수행하지 않고 건너
뛰고, 참일 때 종료

```
1
2
3
4
...
7
8
9
10
```

while - continue문

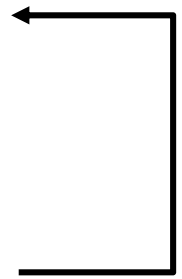
- while 문을 수행할 때 조간식이 맞으면 처음으로 돌아가 조건식을 다시 시작할 때 사용



continue문

- 반복문의 시작 지점으로 이동
- 반복에서 제외하거나 생략하고 싶은 코드가 있을 때 사용

```
a = 0
while a < 10:
    a += 1
    if a % 2 == 0:
        continue
    print(a)
```



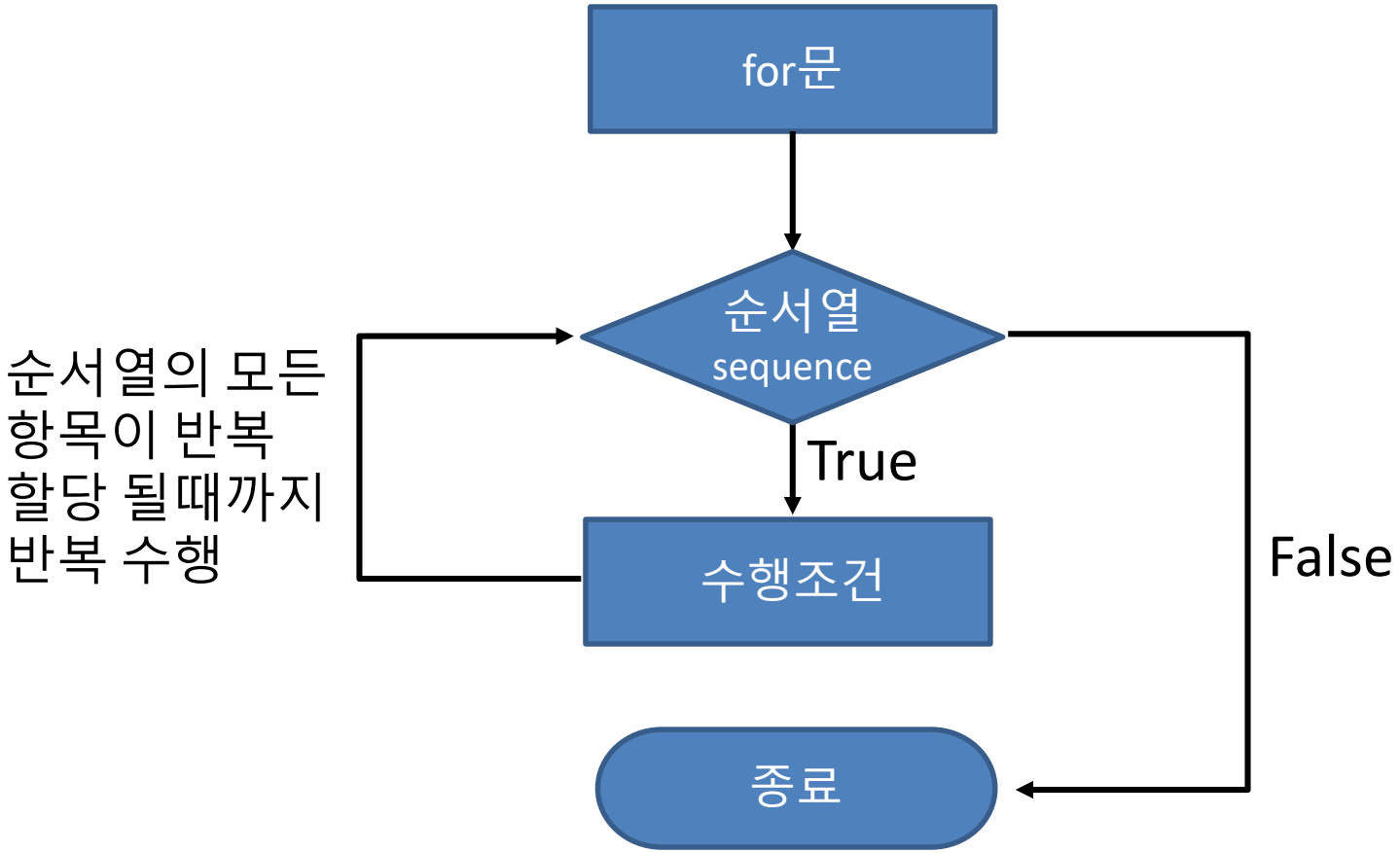
조건을 만족할 때
처음으로 돌아감

조건을 만족하지
않을 때 출력

```
1
3
5
7
9
```

for문

- 값의 범위나 반복 횟수가 정해져 있을 때 주로 사용하는 반복문



for문

- 값의 범위나 반복 횟수가 정해져 있을 때 주로 사용하는 반복문
- 반복 가능 객체 : 문자열, 리스트, 튜플, range, 세트, 딕셔너리

```
for 변수 in 반복 가능 객체:
    반복실행문
```

예)

```
test = [“one”, “two”, “three”]
for i in test:
    print(i)
```

```
one
two
three
```

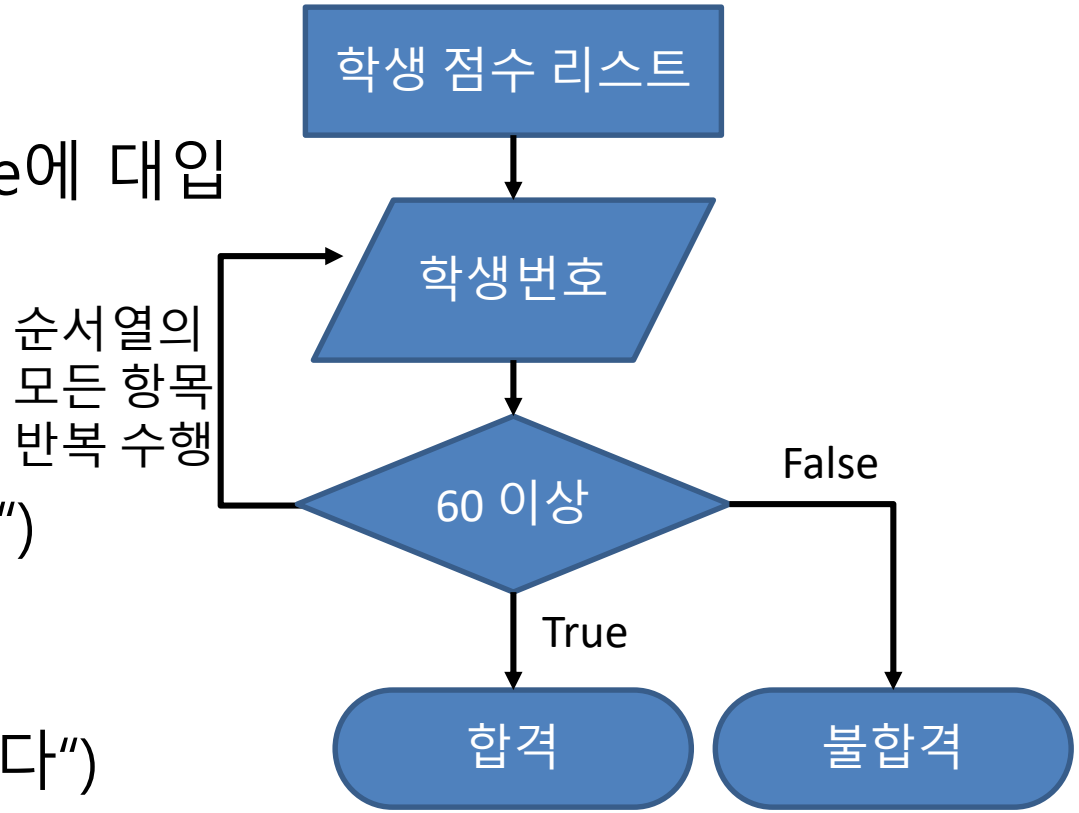
for문

5명의 학생이 시험을 보았는데 시험 점수가 60점을 넘으면 합격이고 60점을 넘지 않으면 불합격

학생들의 시험점수를 차례로 검사해 합격인지 불합격인지 통보해주는 프로그램

for문

```
scores = [90, 25, 67, 45, 80]     # 학생들의 시험 점수 리스트
number = 0    # 학생들에게 붙여줄 번호

for score in scores:  # 점수를 순서대로 score에 대입
    number += 1
    if score >= 60:
        print(f"{number}번 학생은 합격입니다")
    else:
        print(f"{number}번 학생은 불합격입니다")
```



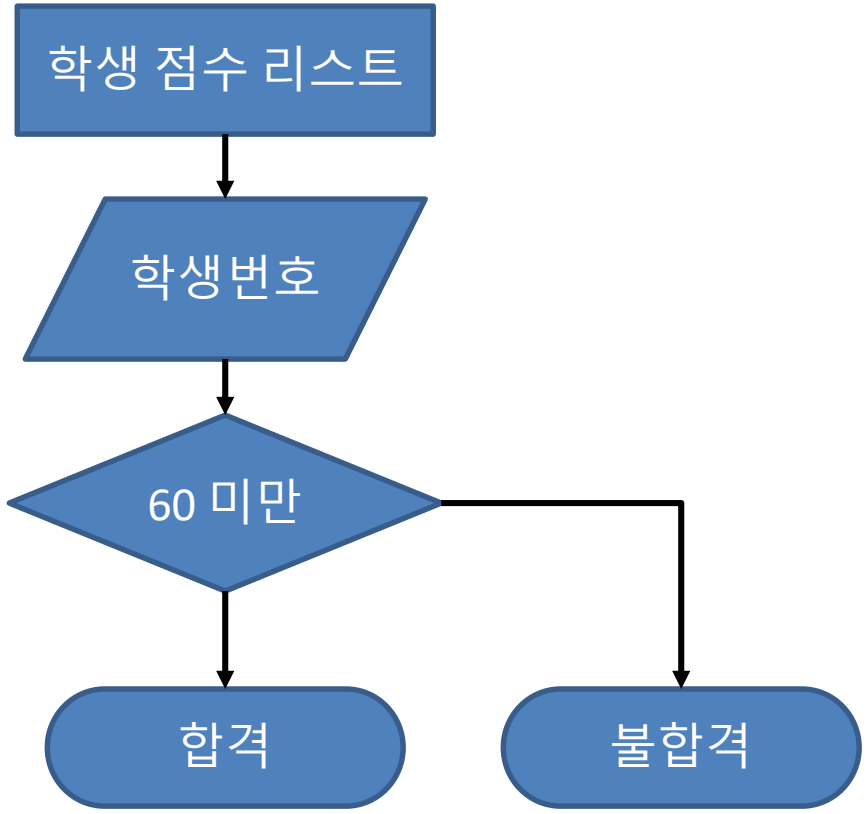
for문

```
scores = [90, 25, 67, 45, 80]     # 학생들의 시험 점수 리스트
number = 0    # 학생들에게 붙여줄 번호

for score in scores:     # 점수를 순서대로 score에 대입
    number += 1
    if score >= 60:
        print(f"{number}번 학생은 합격입니다")
    else:
        print(f"{number}번 학생은 불합격입니다")
```

```
1번 학생은 합격입니다
2번 학생은 불합격입니다
3번 학생은 합격입니다
4번 학생은 불합격입니다
5번 학생은 합격입니다
```

for문 - continue

```
scores = [90, 25, 67, 45, 80]
number = 0

for score in scores:
    number += 1
    if score < 60:
        continue
    else:
        print(f"{number}번 학생은 합격입니다")
```



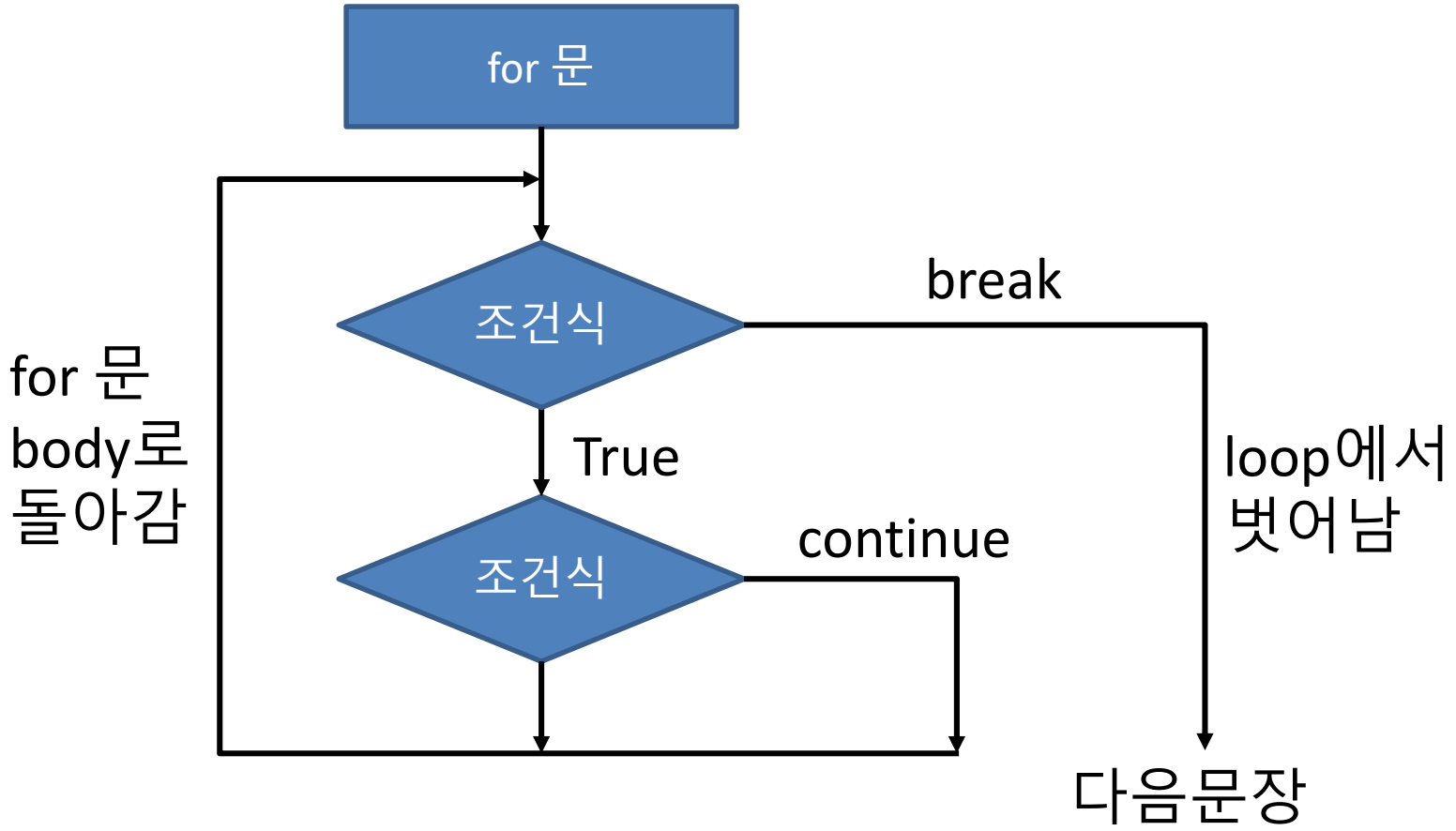
for문 - continue

```
scores = [90, 25, 67, 45, 80]
number = 0

for score in scores:
    number += 1
    if score < 60:
        continue
    else:
        print(f"{number}번 학생은 합격입니다")
```

```
1번 학생은 합격입니다
3번 학생은 합격입니다
5번 학생은 합격입니다
```

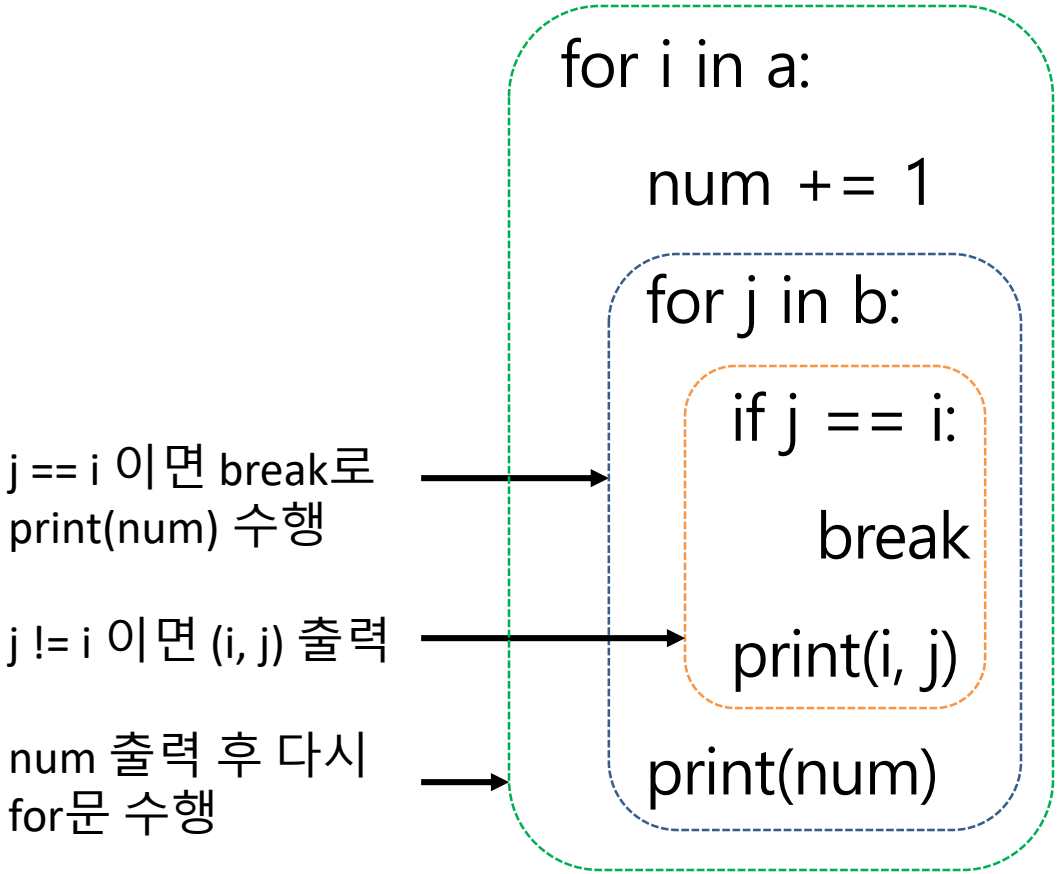
for - break, continue문

- for 문에서 수행문을 처음으로 돌리거나 강제 종료할 때 사용



for문 - break

```
a = [1, 2, 3]
b = [1, 2, 3]
num = 0
```




```
1
2 1
2
3 1
3 2
3
```

while문

while 반복문 수행 구조

```
for i in a:
    num += 1
    for j in b:
        if j == i:
            break
        print(i, j)
    print(num)
```

num	i	j	j == i	for문	출력
1	1	1	참	break	1
1번 for문 종료					

num	i	j	j == i	for문	출력
2	2	1	거짓	수행	2, 1
2	2	2	참	break	1
2번 for문 종료					

num	i	j	j == i	for문	출력
3	3	1	거짓	수행	3, 1
3	3	2	거짓	수행	3, 2
3	3	3	참	break	3
3번 for문 종료					

for문 - range

- 수열을 자동적으로 만들어주는 함수
- range(시작 인덱스, 종료 인덱스, 증감값)

```
for 변수 in range(증감값):
    실행문
```

for문 - range

```
for i in range(10):
    print(i)
```

```
0
1
...
8
9
```

```
for i in range(1, 11):
    print(i, end = " ")
```

```
1 2 3 4 5 6 7 8 9 10
```

```
for i in range(1, 11, 2):
    print(i, end = " ")
```

```
1 3 5 7 9
```

for문 - range

```
tot = 0
for i in range(11):
    tot += i
print(tot)
```

```
55
```

```
# 구구단 출력하기
gugu = 2
 for i in range(1, 10):
    print(f"{gugu} * {i} = {gugu * i}")
```

```
2 * 1 = 2
…
2 * 9 =18
```

연습문제

1부터 1000까지의 자연수 중 3의 배수의 합 구하기

1부터 10까지의 자연수 중 홀수만 2를 곱해서 리스트로 저장하기

계단식으로 별 출력하기

*

**

***

****

연습문제

1부터 1000까지의 자연수 중 3의 배수의 합 구하기

```
result = 0
for i in range(1, 1001):
    if i % 3 == 0:
        result += i
result
```

i 를 3 나눠 나머지가
0이 참이면 result + i 실행

i == 3 일때 result = 0 + 3 = 3
I == 6 일때 result = 6 + 3 = 9
I == 9 일때 result = 9 + 9 = 18
i == 12 일때 result = 12 + 18 = 30
…
i == 996 일때 result = 996 + 1648863 = 165834
i == 999 일때 result = 996 + 165833 = 166833
i == 1000 일때 조건문 False 수행안함

166833

연습문제

1부터 10까지의 자연수 중 홀수만 2를 곱해서 리스트로 저장하기

```
result = [ ]
for i in range(1, 11, 2):
    result.append(i * 2)
result
```

1부터 10까지 2만큼 증가 range – 1 3 5 7 9
1 * 2 = 2
3 * 2 = 4
5 * 2 = 10
7 * 2 = 14
9 * 2 = 18의 append 값을 리스트 [ ]로 출력

[2, 6, 10, 14, 18]

```
result = [ ]
    for i in range(1, 11):
        if i  % 2 == 1:
        result.append(i * 2)
result
```

[2, 6, 10, 14, 18]

연습문제

계단식으로 별 출력하기

*

**

***

****

```
for i in range(4):
    for j in range(4):
        if j <= i:
            print("*", end = " ")
    print( )
```

4번 반복. i 루프는 세로 방향

4번 반복. j 루프는 가로 방향

j <= i 를 만족하면 별 출력. end에 " "를 저정하여 줄바꿈 하지 않음

가로 방향으로 별을 그린 뒤 다음 줄로 넘어감

※ print문은 별도 지정 없으면 다음 줄로 넘어감

함수

- 어떤 동작을 수행하는 코드들의 묶음
- 여러 곳에서 사용되는 코드는 하나의 함수로 사용
- 함수 이름 뒤 ( )안의 매개변수는 함수에 입력으로 전달되는 값을 받는 변수

- 함수의 구조
  - def 함수명(매개변수):
    - 수행할 코드

함수

```
# welcome( )함수 정의
def welcome( ):
    print("Hello")
    print("Nice to meet you")
welcome( ) # 함수 호출
```

매개변수가 없는 함수. ( )가 비어있는 함수는
함수를 정의 후 호출을 할 때만 결과값을 나타냄

```
Hello
Nice to meet you
```

함수

- 매개변수와 인수
  - 매개변수 : 함수에 입력으로 전달된 값을 받는 변수
  - 인수 : 함수를 호출할 때 전달하는 입력 값
  - 함수의 처리과정 : 입력값 → 함수 → 결과값
  - 함수의 사용법 : 결과값 받을 변수 = 함수 이름(입력 인수1, 입력 인수2...)

함수

```
# add( )함수 정의
def add(a, b):          # add( ) 함수 이름이고 a, b는 매개변수
    return a + b


a = 3
b = 4
c = add(a, b)           # add(3, 4)의 반환 값을 c에 대입
print(c)
```

함수

- 디폴트 매개변수
  - 매개변수로 전달되는 인수가 없는 경우에 사용하는 기본값

```
def say_myself(name, old, man = True ):
    print(f"나의 이름은 {name} 입니다")
    print(f"나이는 {old} 살입니다")
    if man:
        print("남자입니다")
    else:
        print("여자입니다")
```

매개변수에 초기값을 미리 설정
매개변수에 들어가는 값이 항상 변하는 것이 아닌 경우 사용 가능
( )가 비어있는 매개변수가 없는 함수로 정의 후 함수를 정의 후 호출을 할 때만 결과값을 나타냄

아무 값도 나타나지 않음

함수

함수 호출

say_myself(“파이썬”, 25)

say_myself(“파이썬”, 25, True)

나의 이름은 파이썬입니다
나이는 25살입니다
남자입니다

say_myself(“파이썬”, 25, False)

나의 이름은 파이썬입니다
나이는 25살입니다
여자입니다

함수

- 디폴트 매개변수의 위치를 맞춰야 함. 다르면 에러 발생

```
def say_myself(name, man = True, old ):
    print(f"나의 이름은 {name} 입니다")
    print(f"나이는 {old} 살입니다")
    if man:
        print("남자입니다")
    else:
        print("여자입니다")
```

함수

함수 호출

say_myself("파이썬", 25)

say_myself("파이썬", 25, True)

say_myself("파이썬", 25, False)

SyntaxError: non-default argument follows default argument

name 변수에는 '파이썬' 이 들어가지만
25를 man 변수와 old 변수 어느 곳에 대입해야 할지 알 수 없기 때문에 에러 발생

함수

- 가변 매개변수 (여러 개의 입력 값을 받는 함수)
  - 매개변수로 전달되는 인수가 없는 경우에 사용하는 기본값
  - 함수 정의 시 매개변수 앞에 *을 붙이면 가변 매개변수가 됨
  - 전달되는 인수는 하나의 튜플 형태로 전달

```
def show(*args):
        print(args)
show(“Python”)                                  (‘Python’)
show(“happy”, “new year”)                       (‘happy’, ‘new year’)
```

함수

```
def add_many(*args):
    result = 0
    for i in args:
        result += i                                  *args에 입력받은 모든 값을 더한다
    return result
result = add_many(1, 2, 3)                           # 1, 2, 3의 합
print(result)                                                    6
result = add_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)    # 1부터 10까지 합
print(result)                                                    55
```

함수

```
def add_mul(choice, *args):
    if choice == "add":          # 매개변수 choice에 'add'를 입력받았을 때
        result = 0
        for i in args:
            result += i          # args에 입력받은 모든 값을 더한다

    elif choice == "mul":        # 매개변수 choice에 'mul'을 입력받았을 때
        result = 1
        for i in args:
            result *= i          # *args에 입력받은 모든 값을 곱한다

    return result
```

함수

```
result = add_mul(“add”, 1, 2, 3, 4, 5)
print(result)
```

15

```
result = add_mul(“mul”, 1, 2, 3, 4, 5)
print(result)
```

120

매개변수 choice에 ‘add’가 입력된 경우 *args에 입력되는 모든 값을 더해서 15를 돌려주고,

'mul ' 이 입력된 경우 *args에 입력되는 모든 값을 곱해서 120을 돌려준다

함수

- 지역변수와 전역변수
  - 지역변수
    - 함수 내부에서 선언한 변수는 함수 내부에서만 사용할 수 있는 변수가 됨
    - 함수 외부에서는 지역변수에 접근할 수 없음
  - 전역변수
    - 함수 외부에서 선언한 변수는 함수 내부에서도 사용 가능

함수 - 지역변수

```
def vartest( ):          # 매개변수가 없는 함수 정의
    test_a = 10          # 함수 안에서의 변수 test_a 선언
    print(test_a)

vartest( )               # 함수 호출                          10

print(test_a)            # 함수 안에서의 지역변수로 함수      NameError : name
                         바깥에서는 정의되어 있지 않음       'test_a' is not defined
```

함수 - 전역변수

```
test_b = 20              # 함수 밖의 변수 test_a

def btest( ):            # btest 함수 선언
    print(test_b)

btest( )                 # 함수 호출                                  20

print(test_b)            # 함수 밖의 전역변수 이기 때문에 출력        20
```

함수

```
def f( ):                  # 매개변수 없는 함수 f( ) 정의
    test_b = 100           # f( ) 함수 안에서의 test_b = 100 선언
    print(test_b)

f( )                       # f( )함수 호출로 함수 안의 test_b 변수 출력      100
print(test_b)              # 이전 함수 밖의 전역변수 호출                    20
```

함수

```
def f( ):
    global test_b        #global 명령어로 전역변수 test_b를 사용

print(test_b)            # global 명령어로 전역변수 test_b 출력        20

f( )                     # global 명령어로 f( )함수 정의가 없어 출력 없음
```

연습문제

몫과 나머지를 구하는 함수를 작성하기

예) 함수(10, 3)

결과값 : 3, 1

연습문제

몫과 나머지를 구하는 함수를 작성하기

예) 함수(10, 3)

결과값 : 3, 1

```
def get_divmod(a, b):
    return a // b, a % b

get_divmod(10, 3)
```

get_divmod( )함수의 결과값은 1개로 a // b, a % b는 튜플의 값으로 각각 출력

(3, 1)

연습문제

주어진 자연수가 홀수인지 짝수인지 판별해주는 함수 작성하기

예) 함수(자연수)

결과값 : True | False

연습문제

주어진 자연수가 홀수인지 짝수인지 판별해주는 함수 작성하기

예) 함수(자연수)

결과값 : True | False

```
def is_odd(num):
    if num % 2 == 1:
        return True

    else:
        return False
is_odd(3)
is_odd(4)
```

is_odd( )함수의 매개변수 num의 조건을 만족하는 것에 따른 홀, 짝수 판별

True

False

함수 실습

- 커피자판기 코드 만들기
- 기능 정의
  - 커피자판기에 돈과 주문할 커피를 전달
  - 주문할 수 있는 커피의 종류와 가격은 다음과 같음
    - 아메리카노 : 1000원
    - 카페라떼 : 1500원
    - 카푸치노 : 2000원
  - 없는 커피를 주문할 경우 입력한 돈을 그대로 반환
  - 구매 금액이 부족하면 입력한 돈을 그대로 반환
  - 정상 주문이면 주문한 커피와 잔돈을 반환

함수 실습

```
def coffee_machine(money, pick):
    print(f"{money}원에 {pick}를 선택하셨습니다")

    menu = {"아메리카노" : 1000, "카페라떼" : 1500, "카푸치노" : 2000}

    if pick not in menu:
        print(f"{pick}는 없는 메뉴입니다")
        return money

    elif menu[pick] > money:
        print(f"{pick}는 {menu[pick]}원입니다")
        print(f"돈이 {menu[pick] - money}원 부족합니다")
        return money

    else:
        print(f"{pick}는 {menu[pick]}원입니다")
        print(f"잔돈은 {money - menu[pick]}원입니다")
        return pick, money - menu[pick]
```

coffee_machine( )함수에 money, pick 변수 정의

coffee_machine( )함수에 money, pick 변수 정의

조건식 menu에 pick 변수가 없을 때 출력 후 money return

조건식 menu에 pick 변수를 만족하나 money가 부족할 경우 money return

조건식 menu에 pick 변수를 만족하고 money도 만족할 pick와 money-menu 차액 return

함수 실습

coffee_machine(pick = “마끼아또”,
money = 1500)

1500원에 마기아또를 선택하셨습니다
마끼아또는 없는 메뉴입니다
1500

coffee_machine(pick = “아메리카노”,
money = 500)

500원에 아메리카노를 선택하셨습니다
아메니카노는 1000원입니다
500

coffee_machine(pick = “카페라떼”,
money = 2000)

2000원에 카페라떼를 선택하셨습니다
카페라떼는 1500원입니다
('카페라떼', 500)

데이터 입출력

- 사용자 입력
  - 사용자가 값을 입력하게 하는 것
  - input은 입력되는 모든 데이터를 문자열로 취급함
  - 프롬프트 값을 띄워서 사용자 입력 받기

input

```
a = input( )
사용자 입력
print(a)
```

```
input("질문 내용")
number = input("질문 내용")
print(number)
```

데이터 입출력

```
# input
a = input( )
사용자 입력          # 숫자, 문자열, 리스트의 입력한 자료형을 출력
print(a)

number = input("숫자를 입력하세요 : ")
사용자 입력
print(number)
print(type(number))
```

```
python
python

숫자를 입력하세요 : 3
3
<class 'str'>
```

파일 입출력 - 파일생성

- 파일 입출력
  - 파일 입력 : 파일의 내용을 읽어들이는 것
  - 파일 출력 : 파일에 새로운 내용을 추가하거나 새 파일을 생성하는 것

- 파일 생성
  - 파일 객체 = open(파일이름, 파일열기모드)

```
f = open("new_file.txt", "w")          # new_file 이름의 txt 파일 생성
f.close( )
```

파일 출력

- 파일 출력

```
f = open("new_file.txt", "w")

for i in range(1, 11):                  # 1부터 10까지 i 에 대입
    f.write(str(i) + "₩n")              # str( i )를 줄바꿈한 값을 파일 객체 f에 write

f.close( )
```

파일 열기

- 파일 열기 모드
  - r(읽기 모드)
    - 파일을 읽기만 할 때
    - 파일이 없으면 error
  - w(쓰기 모드)
    - 파일에 내용을 쓸 때
    - 파일이 없으면 새로 만듦
    - 기존 파일에 있던 데이터를 완전히 지우고 다시 쓴다
  - a(추가 모드)
    - 파일의 마지막에 새로운 내용을 추가할 때
    - 파일이 없으면 새로 만듦

클래스

- 객체를 만드는 도구
- 클래스를 통해 여러 개의 객체를 만들 수 있음
- 동일한 클래스에서도 서로 다른 값을 가진 객체가 만들어질 수 있음

```
class 클래스 이름( ):
    def 메서드(self):
        코드
인스턴스 = 클래스( )
인스턴스.메서드( )        #호출
```

클래스

- 클래스의 구성
  - 클래스는 객체가 가져야 할 구성요소를 모두 가지고 있어야 함
    - 값 : 이름, 나이, 연락처, 주소 등(변수)
    - 기능 : 잔다, 먹는다, 공부한다, 달린다 등(함수)

- 인스턴스 변수와 인스턴스 메소드
  - 인스턴스 변수 : 클래스를 기반으로 만들어지는 모든 객체들이 각각 따로 저장하는 변수
    - 모든 인스턴스 변수는 self라는 키워드를 붙임
  - 인스턴스 메소드
    - 인스턴스 변수값에 따라서 각 객체마다 다르게 동작
      - 첫 번째 매개변수로 self를 추가

클래스

```
result = 0
def call_add(num):
    global result                # global 명령어로 result를 전역변수로 사용
    result = result + num
    return result


call_add(3)                      # call_add함수는 매개변수 num에 받은 값을 이전에 계산한 결과값에 더한 후 값 반환
call_add(4)
call_add(4)
```

3

7

11

클래스 – 2개의 계산기 만들기

```
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, num):
        self.result = self.result + num
        return self.result

cal1 = Calculator( )
cal2 = Calculator( )
```

__init__ 함수는 클래스에 ( )를 붙여서 인스턴스를 만들 때 호출되는 메서드로 다른 함수에서도 self.변수명으로 통해 이 변수에 접근할 수 있음

class를 선언하고 init( )함수를 사용하여 클래스에서 사용할 변수들을 초기화 설정

add( )함수는 매개변수 num에 받은 값을 이전에 계산한 결과값에 더한 후 값 반환

```
cal1.add(3)   3        cal2.add(3)   3
cal1.add(4)   7        cal2.add(7)   10
```

인스턴스 변수와 인스턴스 메소드

- 인스턴스 변수 : 클래스를 기반으로 만들어지는 모든 객체들이 각각 따로 저장하는 변수
  - 모든 인스턴스 변수는 self라는 키워드를 붙임
  - 클래스의 속성을 나타내면서 각각의 인스턴스 마다 다른 값을 갖게 할 용도로 사용

- 인스턴스 메소드
  - 인스턴스 변수값에 따라서 각 객체마다 다르게 동작
    - 첫 번째 매개변수로 self를 추가

클래스

```
# 예시1. Person 클래스를 정의
class Person:
    def who_am_i(self, name, age, tel, address):
        # 인스턴스 메소드 who_am_i
        # 모든 Person클래스의 객체는 who_am_i( ) 메소드를 호출 가능
        # self를 제외한 나머지 매개변수에 실제로 사용될 데이터가 전달

        self.name = name
        # 인스턴스 변수 name
        # = 오른쪽에 있는 name은 매개변수의 name
        # who_am_i( ) 메소드를 호출할 때 전달된 name이 객체의 name이 됨
```

클래스

```
self.age = age
self.tel = tel
self.address = address

boy = Person( )    # 객체 boy를 생성

boy.who_am_i("홍길동", 25, "123-456", "서울")      # 객체 boy에 who_am_i 함수의 변수 입력
```

boy.name	# 객체 boy의 name	'홍길동'
boy.age	age	25
boy.tel	tel	'123-456'
boy.address	address	'서울'

클래스

```
# 예시2.
class Computer:
    def set_spec(self, cpu, ram, vga, ssd):          # set_spec 메서드의 매개변수
        self.cpu = cpu
        self.ram = ram                               # 메서드의 수행문
        self.vga = vga
        self.ssd = ssd

    def hardware_info(self):                         # 메서드의 매개변수에 대한 정의
        print(f"CPU = {self.cpu}")                   # 메서드를 호출하면 실행
        print(f"RAM = {self.ram}")
        print(f"VGA = {self.vga}")
        print(f"SSD = {self.ssd}")
```

클래스

```
desktop = Computer( )
desktop.set_spec("i7", ""16GB", "GTX3060", "512GB")
desktop.hardware_info( )
```

```
CUP = i7
RAM = 16GB
VGA = GTX3060
SSD = 512GB
```

```
notebook = Computer( )
notebook.set_spec("i5", ""8GB", "MX300", "256GB")
notebook.hardware_info( )
```

```
CUP = i5
RAM = 8GB
VGA = MX300
SSD = 256GB
```

모듈

- 함수나 변수 또는 클래스를 모아 둔 파일
- 다른 파이썬 프로그램에서 불러와 사용할 수 있게 만든 파이썬 파일

- 모듈 사용
  - import 모듈
  - from 모듈 import 함수
  - from 모듈 import 함수1, 함수2
  - from 모듈 import *

모듈

```
import add_sub_module
print(add_sub_module.add(3, 4))          7
print(add_sub_module.sub(4, 2))          2

from add_sub_module import add, sub
add(3, 4)                                7
sub(4, 3)                                1

add                                      <function add_sub_module.add(a, b)>
```

모듈

```
import converter
miles = converter.kilometer_to_miles(160)
print(f"160km = {miles} miles")                    160km = 99.41936miles
pounds = converter.gram_to_pounds(1000)
print(pounds)                                      2.20462

from converter import *
miles = kilometer_to_miles(140)
print(miles)                                       86.99194
pounds = converter.gram_to_pounds(1000)
print(pounds)                                      0.220462
```

모듈

• 별명 사용하기

```
import converter as cvt
miles = cvt.kilometer_to_miles(150)
print(miles)                                    93.20565

pounds = cvt.gram_to_pounds(1000)
print(pounds)                                   2.20462

from converter import kilometer_to_miles as k2m
miles = k2m(150)
print(miles)                                    93.20565
```

모듈

- 표준모듈
  - 파이썬에 기본적으로 설치되어 있는 모듈
  - 별도의 설치없이 import 사용 가능

- math
  - 수학과 관련된 값과 함수를 제공

모듈

```
import math
# 원주율
math.pi                                  3.141592653589793

# 올림과 내림
print(math.ceil(-1.9)) # 올림             -1
print(math.floor(-1.9)) # 내림            -2
```

모듈

```
# 소수점 이하 절사
print(math.trunc(-1.9)) # 절사                    -1
print(math.floor(-1.9))                          -2

# 제곱근
math.sqrt(25)                                    5.0
```

모듈

- random
  - 난수 생성 모듈

```
import random
random.randint(1, 10)                    5
#randint( ) : 특정 범위에 속한 정수 중에서 하나를
임의로 생성
random.randrange(10)                     0
random.randrange(1, 10)                  9
random.randrange(1, 10, 2)               3
```

모듈

- random.random( )

#random( ) : 0이상 1미만 범위에서 임의의 실수를 생성

0% 이상 100% 미만으로 확률을 처리할 때에도 사용

random.random( ) 0.05026379561347016 4

#50% 확률롸 안녕하세요를 출력하는 코드

```
if random.random( ) > 0.5:
    print(“안녕하세요”)
```

안녕하세요

또는 빈칸

모듈

- choice( )
  - 전달된 시퀀스 자료형에 속한 요소 중에서 하나를 임의로 반환

```
seasons = [“spring”, “summer”, “fall”, “winter”]
random.choice(seasons)
```

- sample( )
  - 전달된 시퀀스 자료형에 속한 요소 중 지정된 개수의 요소를 임의로 반환
  - 반환 결과는 리스트 자료형
  - 중복없이 선택

모듈

```
random.sample(range(1, 46), 6)                    [27, 23, 40, 8, 37, 41]
sorted(random.sample(range(1, 46), 6))            [9, 12, 13, 18, 24, 40]
```

- suffle( )
  - 임의로 섞는 것
  - 전달된 시퀀스 자료형에 속한 요소의 순서를 임의로 조정하여 다시 재배치
  - 실제로 전달된 시퀀스 자료형의 순서가 재배치됨
  - str이나 튜플 자료형을 전달하면 에러

```
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
my_list                                           [1, 3, 5, 4, 2]
```