

# 으뜸 파이썬



## 3장 제어문

## 3.1 순차문sequential statements

- 순차적 구조
  - 먼저 나타나는 코드가 먼저 실행되는 구조

코드 3-1 : 순차적 실행 구조를 이용한 변수의 덧셈

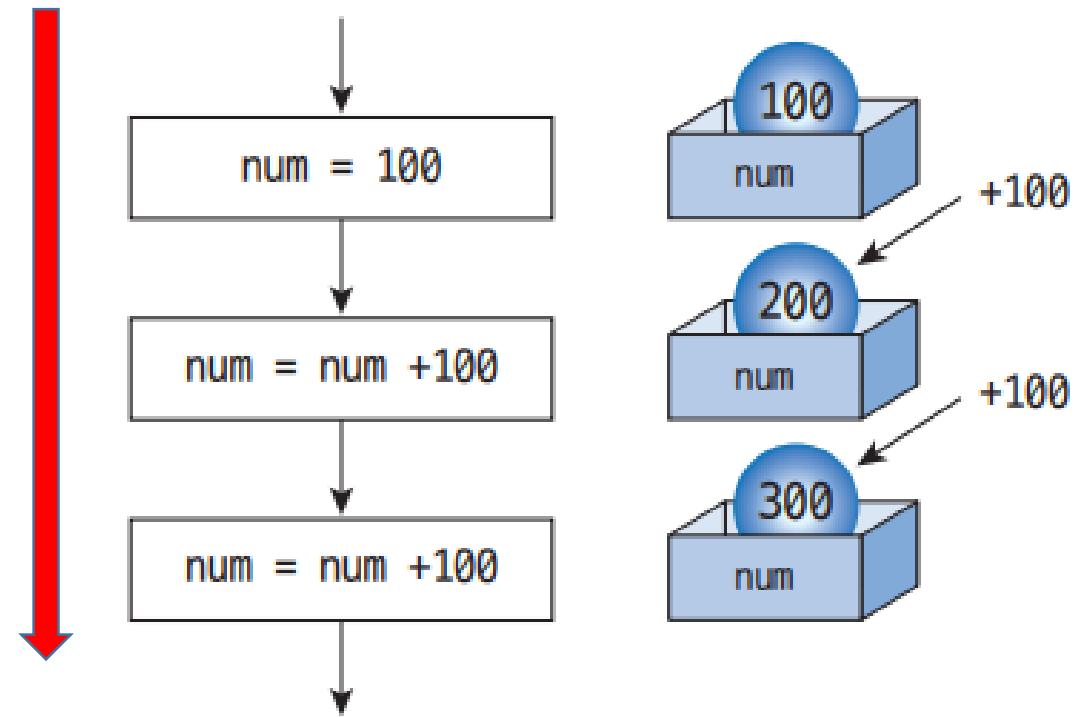
seq\_test.py



```
num = 100
print('num = ', num)      # 100이 출력됨
num = num + 100
print('num = ', num)      # num에 100이 더해져 200이 출력됨
num = num + 100
print('num = ', num)      # num에 다시 100이 더해져 300이 출력됨
```

실행결과

```
num = 100
num = 200
num = 300
```



[그림 3-1] [코드 3-1]의 순차적인 실행 순서와 변수 값의 변화

# 순차문 sequential statements 이외의 흐름문 flow statements

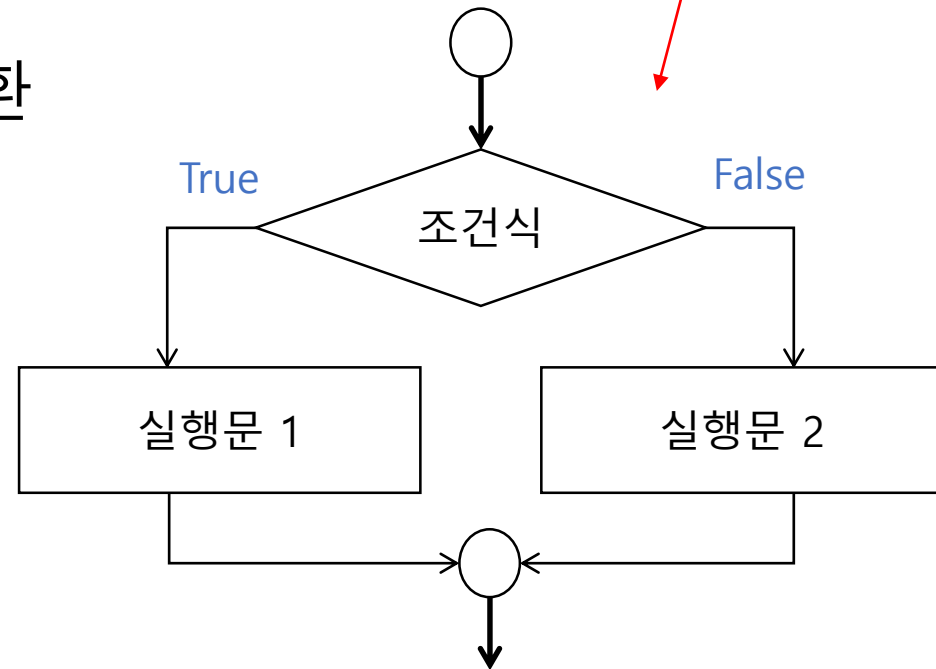
- 제어문 control statements
  - 프로그램의 흐름을 제어하는 역할
  - 조건문 conditional statements
    - if 문, if-else 문
  - 반복문
    - for 문, while 문
  - 반복문의 흐름 변경
    - break, continue

## 3.2 if 조건문

- 조건문 **conditional statements**

- 실행을 달리하는 여러 개의 실행문이 있음
- 특정한 조건에 따라서 실행됨
- 조건식은 True 혹은 False를 반환

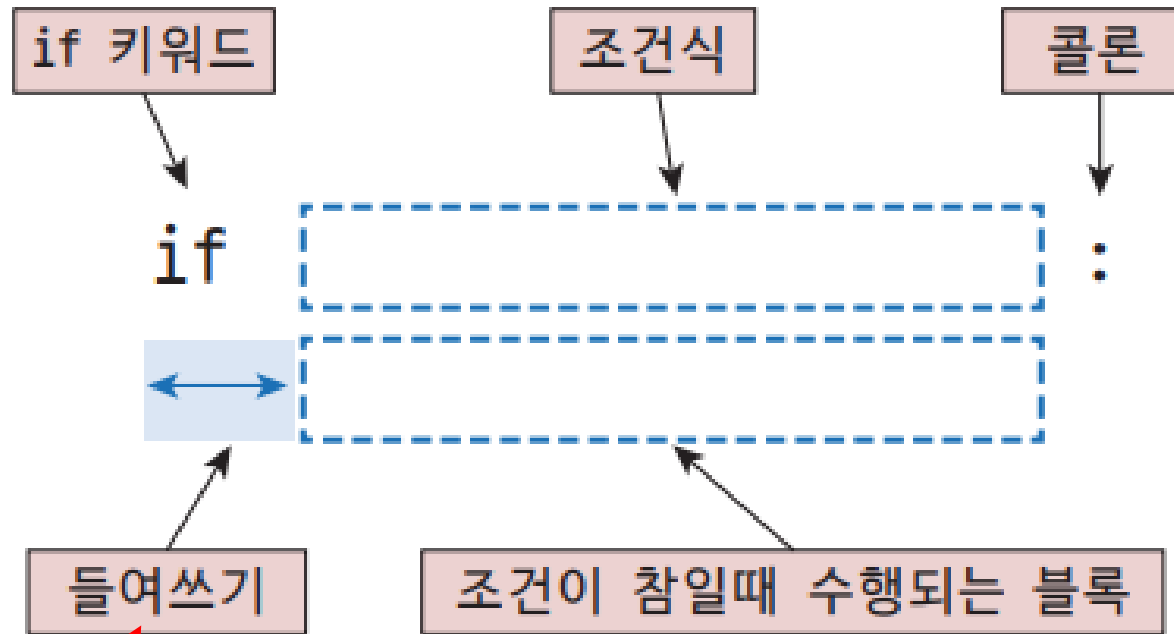
흐름도 :  
이와 같은 작업의 흐름을 나타내는 그림을  
흐름도라고 한다.



# 조건문의 구체적 상황

- 상황 1 : 나이가 20세 미만이면 '청소년 할인'을 출력하는 기능
- 상황 2 : 1000 걸음 이상을 걸으면 '목표 달성'을 출력하는 기능
- 상황 3 : 시간이 12시가 안되면 '오전입니다', 12시 이후이면 '오후입니다'를 출력하는 기능

# if 조건문의 사용법



[그림 3-3] if 조건문의 사용법

파이썬은 들여쓰기가 아주 중요하다

상황 1 :  
나이(age)가 20세 미만이면 '청소년  
할인'을 출력



```
if age < 20 :  
    print('청소년 할인')
```

상황 2 :  
걸음(walk\_count)이 1000 이상이면  
'목표 달성' 출력



```
if walk_count >= 1000 :  
    print('목표 달성')
```

- (상황 1) - 콜론(:)앞에 나타나는 조건문 절에서 < 연산자를 이용해 나이(age)가 20세 미만인 경우에만 print('청소년 할인')이라는 코드를 실행
- (상황 2) - 조건문 절에서 >= 연산자를 이용해 걸음(walk\_count)이 1,000 이상이 되면 print('목표 달성')이라는 코드를 실행



# 상황 1

코드 3-2 : if 조건문을 이용한 출력기능(조건을 만족하는 경우)

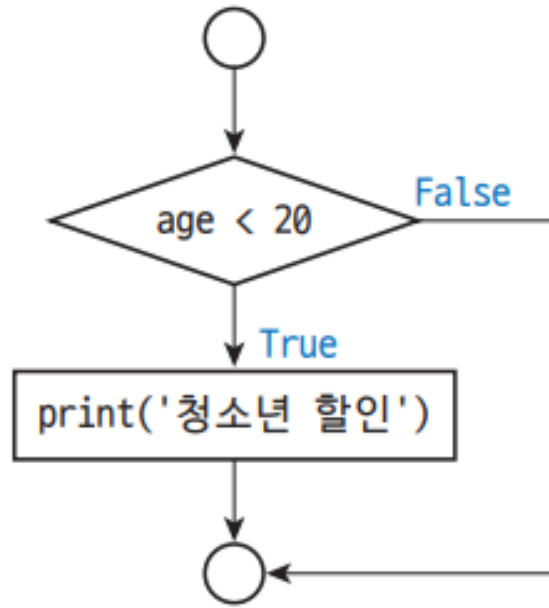
if\_youth\_discount.py

```
age = 18                # age가 20 미만의 값

if age < 20:            # age < 20 조건식의 결과는 True임
    print('청소년 할인')
```

실행결과

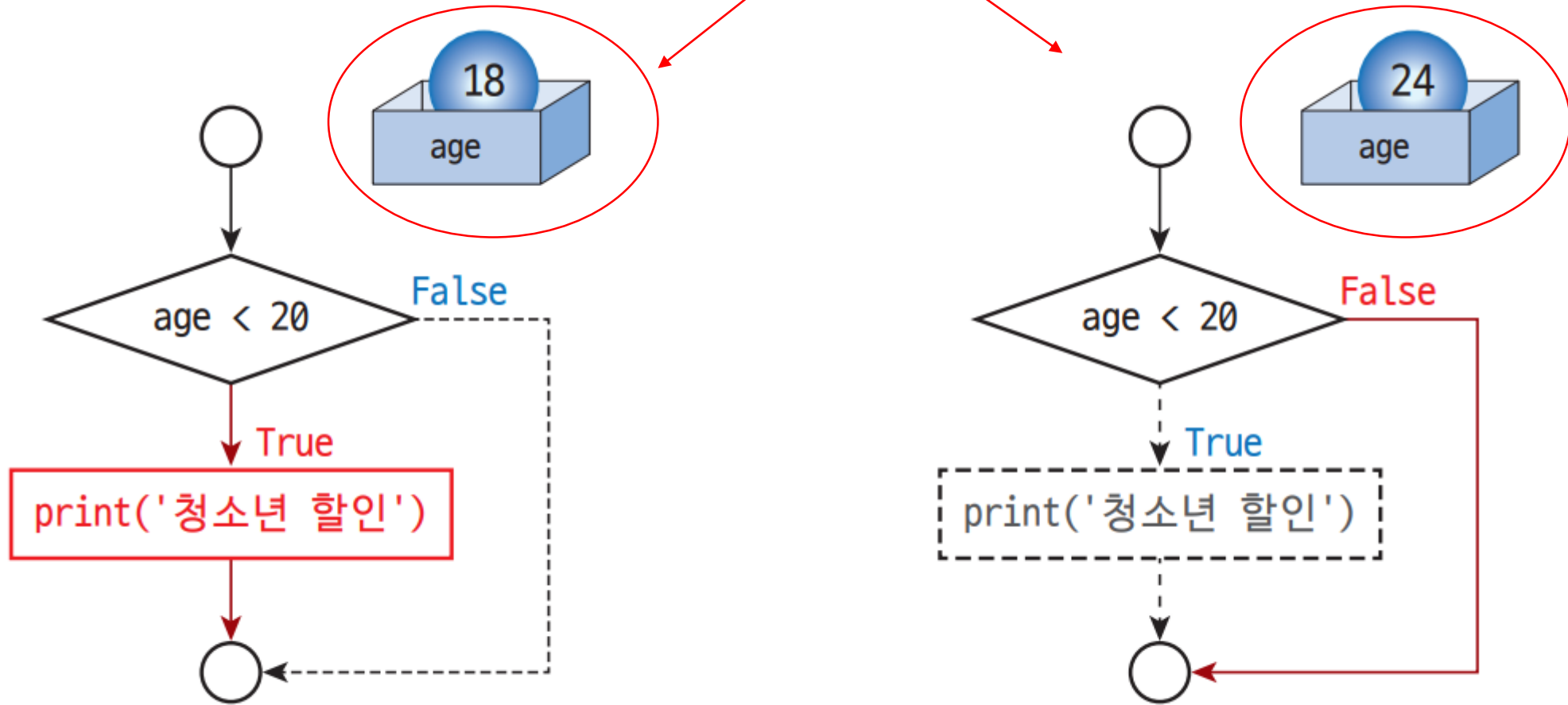
청소년 할인



- age값이 18인 경우  $age < 20$  이라는 조건문이 참(True)이 되므로 '청소년 할인' 이 화면에 출력
- age 값이 20이상이면 아무런 출력을 보내지 않음

[그림 3-5] 흐름도로 표현한 [코드 3-2]의 실행 구조

age 변수의 값에 따라 다른 흐름으로 이동



[그림 3-6] 변수 age의 변화에 따라 제어되는 [코드 3-2]의 실행 흐름

코드 3-3 : if 조건문을 이용한 출력기능(조건을 만족하지 않는 경우)

if\_youth\_discount.py

```
age = 24                # age가 20 이상의 값

if age < 20:            # age < 20 조건식의 결과는 False임
    print('청소년 할인')
```

실행결과

age 변수의 값이 24이면 아무런 출력도 없음

# 상황 2

- 걸음걸이 수(walk\_count)가 1000 이상인 경우에 '목표달성'을 출력하는 프로그램

코드 3-4 : if 조건문을 이용한 출력기능(조건을 만족하는 경우)

```
if_walk_count.py
```

```
walk_count = 1200
```

```
if walk_count >= 1000:    # walk_count >= 1000 조건식의 결과는 True임  
    print('목표 달성')
```

실행결과

목표 달성

- 첫 줄을 walk\_count = 800와 같이 수정한다면 아무런 출력결과를 볼 수 없음

```
walk_count = 800
```

실행결과

if 문의 조건을 만족하지 않을 경우 아무것도 출력되지 않음



### LAB 3-1 : if 문의 사용법

1. 게임 사용자의 게임점수(game\_score)가 1000점 이상이면 '당신은 고수입니다'를 출력하는 프로그램을 if 문을 이용하여 작성하시오. 이때 다음과 같이 game\_score값을 화면에 출력하여라. game\_score에 800점, 1300점을 각각 입력하여 출력문을 확인하시오.

```
game_score = 800
```

혹은

```
game_score = 1300
```

당신은 고수입니다

2. num\_a와 num\_b에 할당된 값이 같으면 '두 값이 일치합니다.'를 출력하는 프로그램을 if 문을 이용하여 작성하시오. num\_a와 num\_b에 각각 100과 200이 할당되어 있는 경우와 num\_a와 num\_b에 300과 300이 할당되어 있는 경우에 대하여 각각 코드를 작성하고 출력문을 확인하시오.

```
num_a = 100, num_b = 200
```

혹은

```
num_a = 300, num_b = 300
```

두 값이 일치합니다.

## 3.2.1 조건문과 블록

- 블록 **block**

- 어떤 조건을 만족하는 경우에 특정한 코드를 선택적으로 실행하는 구조
- 이때 실행될 코드 덩어리
- 블록은 반드시 들여쓰기를 해야한다

코드 3-5 : 들여쓰기 없는 print() 문

if\_youth\_error.py

```
age = 18
```

```
if age < 20:
```

```
print('청소년 할인')      # 들여쓰기 없는 print()문
```

들여쓰기(indentation) 블록이 필요하다는 의미임

실행결과

IndentationError: expected an indented block



# 파이썬의 들여쓰기

- 파이썬은 들여쓰기가 매우 중요한 의미를 가지는 프로그래밍 언어
  - C/C++이나 Java, Pascal등 전통적인 프로그래밍 언어와 다른 특징

[들여쓰기 코드 1] 조건을 만족하는 경우	[들여쓰기 코드 2] 조건을 만족하지 않는 경우
<pre>age = 18 if age &lt; 20:     print('청소년 할인') print('입장을 환영합니다')</pre>	<pre>age = 24 if age &lt; 20:     print('청소년 할인') print('입장을 환영합니다')</pre>
수행결과 - if문 내부와 외부 print문이 수행됨	수행결과 - if문 외부의 print문만 수행됨
청소년 할인 입장을 환영합니다	입장을 환영합니다

[들여쓰기 코드 3] 조건을 만족하는 경우	[들여쓰기 코드 4] 조건을 만족하지 않는 경우
<pre> age = 18 if age &lt; 20:     print('나이', age)     print('청소년 환영')     print('청소년 할인') </pre>	<pre> age = 24 if age &lt; 20:     print('나이', age)     print('청소년 환영')     print('청소년 할인') </pre>
<p>수행결과</p> <ul style="list-style-type: none"> <li>- 들여쓰기 블록 전체가 수행됨</li> </ul>	<p>수행결과</p> <ul style="list-style-type: none"> <li>- 들여쓰기 블록 전체가 수행되지 않음</li> </ul>
<pre> 나이 18 청소년 환영 청소년 할인 </pre>	

- **블록**은 흔히 **코드 블록**이라고도 함
- 소스 코드에서 함께 묶을 수 있는 코드의 덩어리를 말한다
- 파이썬은 if문 다음에 :(콜론)이 나오면 다음에 들여쓰기 코드 블록이 나와야 하며 else, elif, for, while, def, class 등에서도 코드 블록이 사용됨

- [들여쓰기 코드 5]와 같이 동일한 블록에 대해 들여쓰기의 칸 수가 일정하지 않으면 "IndentationError: unexpected indent" 라는 들여쓰기 오류가 발생
- 동일한 코드 블록에서는 들여쓰기의 칸 수를 반드시 일치시켜야 함
- 스페이스 `space` 4칸을 권장

들여쓰기 코드 5 : 들여쓰기가 잘못된 경우

```
age = 18
if age < 20:
    print('나이', age)
    print('청소년 환영')
print('청소년 할인')
```

수행결과

IndentationError: unexpected indent 오류 발생

# 블록의 규칙

- 조건문에서 if 조건에 따라 실행 여부가 결정되는 코드 집합을 **블록** Block이라고 한다



## NOTE : 들여쓰기 블록의 규칙

반드시 콜론 다음 줄에 써야 한다: 들여쓰기 칸수는 몇 칸이든 상관없지만, 파이썬에서는 4칸을 권장한다. 대부분의 파이썬 **통합 개발환경** IDE: Integrated Development Environments에서는 자동 들여쓰기 기능을 제공한다. 또한, 들여쓰기를 할 때 탭보다는 스페이스 키를 권장한다. 블록은 여러 줄로 작성할 수 있다. 단, 여러 줄 들여쓰기를 하는 경우 들여쓰기 칸 수가 모두 같아야 한다.



## PEP 8 -- Style Guide for Python Code

PEP:	8
Title:	Style Guide for Python Code
Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

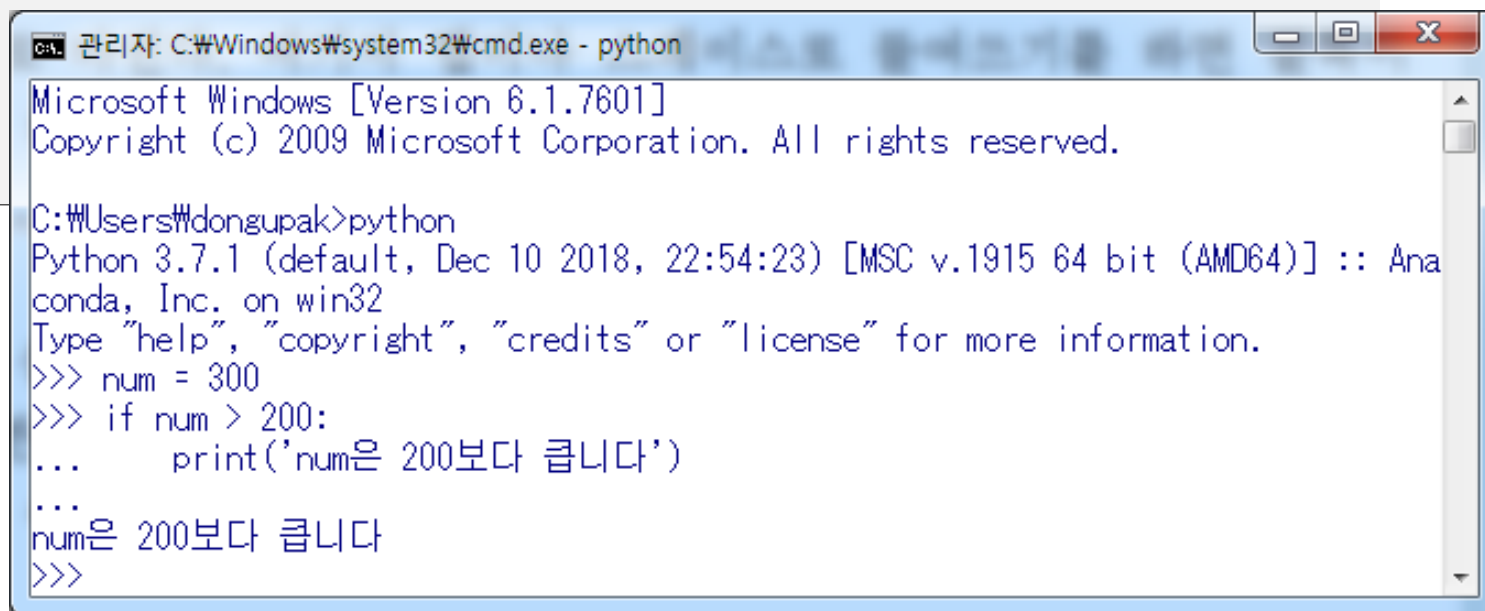
파이썬 소프트웨어 재단 홈페이지에는 파이썬의 여러가지 기능을 규정하는 파이썬 확장 제안서 PEP: Python Enhanced Proposals 문서가 있다. 이 PEP 문서는 파이썬 사용자와 개발자 공동체의 합의된 내용을 담고 있는데 이 중에서 PEP 8 문서는 파이썬의 코딩 스타일을 규정하고 있다. 이 문서의 내용을 살펴보면 다음과 같이 파이썬에서 탭을 사용할지 스페이스를 사용할지에 대한 합의된 내용이 나타나 있다.

그 중에서 탭과 스페이스에 관한 규정이 있는데, 요약하면 들여쓰기에서 스페이스 키를 사용하는 것을 더 권장하며, 탭은 이미 탭으로 들여쓰기가 된 코드와 일관성을 유지하기 위해서만 제한적으로 사용하는 것이 낫다고 규정되어 있다. 뒷 장에서 자세히 설명을 더 하겠지만 코딩시에 일관성 있는 방식으로 코딩하는 습관을 가지는 것은 개발자에게 매우 중요한 덕목이다.

## 3.2.2 대화형 모드의 블록

대화창 실습 : 대화형 모드를 통한 블록과 들여쓰기 실습

```
>>> num = 300
>>> if num > 200:
...     print('num은 200보다 큼니다.')
...
num은 200보다 큼니다.
>>>
```

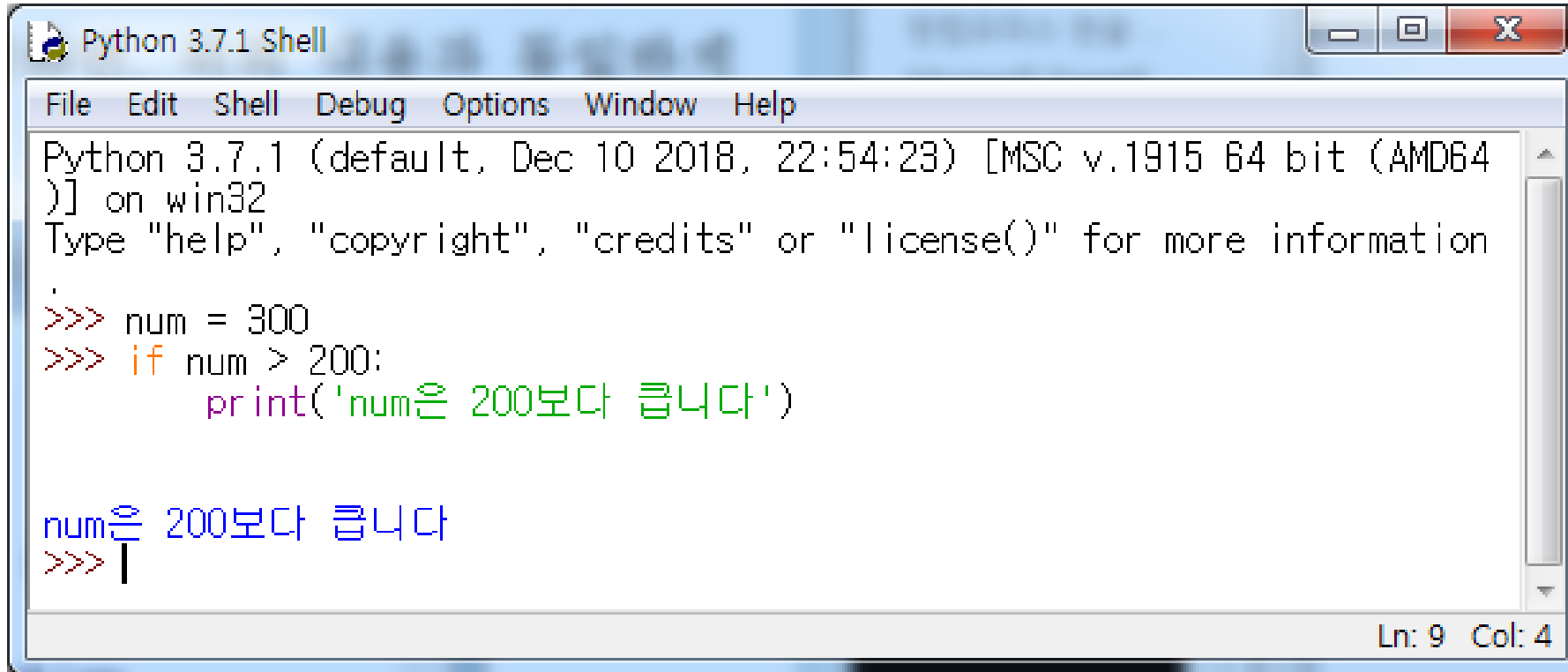


The screenshot shows a Windows command prompt window titled "관리자: C:\Windows\system32\cmd.exe - python". The window displays the output of running a Python script. The first part shows the Microsoft Windows version and copyright information. The second part shows the Python version (3.7.1) and the Anaconda environment. The third part shows the execution of the Python code from the previous block, which prints "num은 200보다 큼니다."

```
관리자: C:\Windows\system32\cmd.exe - python
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\dongupak>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> num = 300
>>> if num > 200:
...     print('num은 200보다 큼니다')
...
num은 200보다 큼니다
>>>
```

# 파이썬 IDLE에서 파이썬 셸 수행



```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information
>>> num = 300
>>> if num > 200:
    print('num은 200보다 큼니다')

num은 200보다 큼니다
>>> |
```

Ln: 9 Col: 4



# 3의 배수 판단

코드 3-6 : 3의 배수를 판단하기 위한 모듈로 연산과 조건문

if\_modulo1.py

```
number = int(input('정수를 입력하세요 : '))    # 입력값을 정수형으로 변환
if number % 3 == 0:                             # 모듈로 3의 값이 0이면 3의 배수임
    print(number, '은(는) 3의 배수입니다.')
```

실행결과

정수를 입력하세요 : 15  
15 은(는) 3의 배수입니다.

실행결과

정수를 입력하세요 : 16

# 3과 5의 배수 판단

코드 3-7 : 3과 5의 배수를 판단하기 위한 모듈로 연산과 and 조건문

if\_modulo2.py

```
number = int(input('정수를 입력하세요 : '))  
if number % 3 == 0 and (number % 5) == 0:  
    print(number, '은(는) 3의 배수이면서 5의 배수입니다.')
```

오류 : 책에는 3으로 표기됨

실행결과

정수를 입력하세요 : 15

15 은(는) 3의 배수이면서 5의 배수입니다.



## LAB 3-2 : 변수와 if 조건식 사용하기

1. 1에서 100 사이의 임의의 정수  $n$ 을 입력받아서 1)  $n$ 을 화면에 출력한 후, 2)  $n$ 이 짝수이면 "...은(는) 짝수입니다."를 다음과 같이 출력하는 프로그램을 작성하여라.

정수를 입력하세요 : 50

$n = 50$

50 은(는) 짝수입니다.

또는

정수를 입력하세요 : 75

$n = 75$

2. -100에서 100 사이의 임의의 정수  $x$ 를 입력받아서 1)  $x$ 를 화면에 출력한 후, 2)  $x$ 가 0보다 큰 정수이면 "...은(는) 자연수입니다."를 다음과 같이 출력하는 프로그램을 작성하여라. 그렇지 않을 경우  $x = -10$ 과 같이  $x$ 를 단순 출력하여라.

정수를 입력하세요 : 50

$x = 50$

50 은(는) 자연수입니다.

또는

정수를 입력하세요 : -10

$x = -10$

## 3.3 if-else 조건문

### 상황 3 : 24시 체계->12시 체계

코드 3-8 : if 문을 이용한 '오전' 혹은 '오후'의 출력기능

```
if_hour_test.py
```

```
hour = 10
```

```
if hour < 12:
```

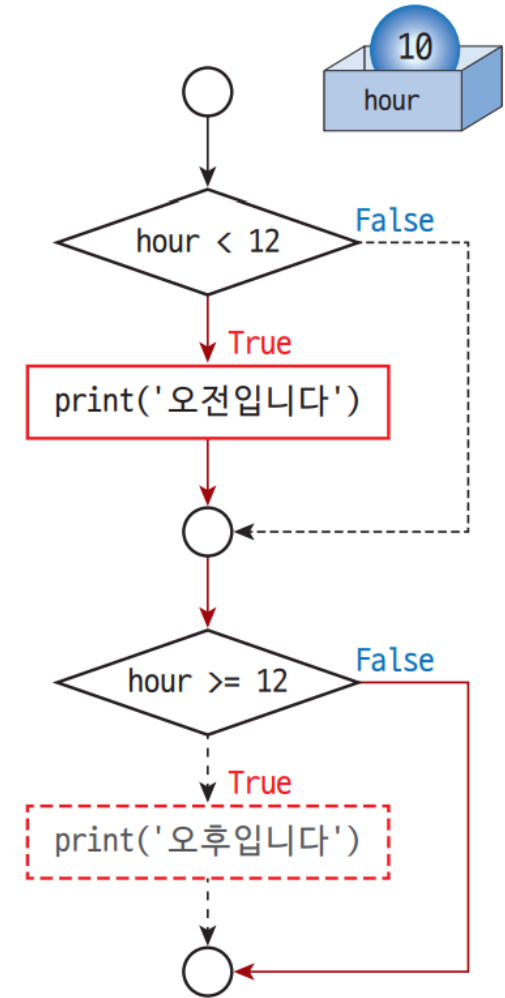
```
    print('오전입니다.')
```

```
if hour >= 12:
```

```
    print('오후입니다.')
```

실행결과

오전입니다.



[그림 3-10] hour 값이 10일 때 코드 if\_hour\_test.py의 흐름도

# if-else 문을 이용한 출력 : 배타적 관계

코드 3-9 : if-else 문을 이용한 '오전' 혹은 '오후'의 출력 기능

if\_else\_hour\_test.py

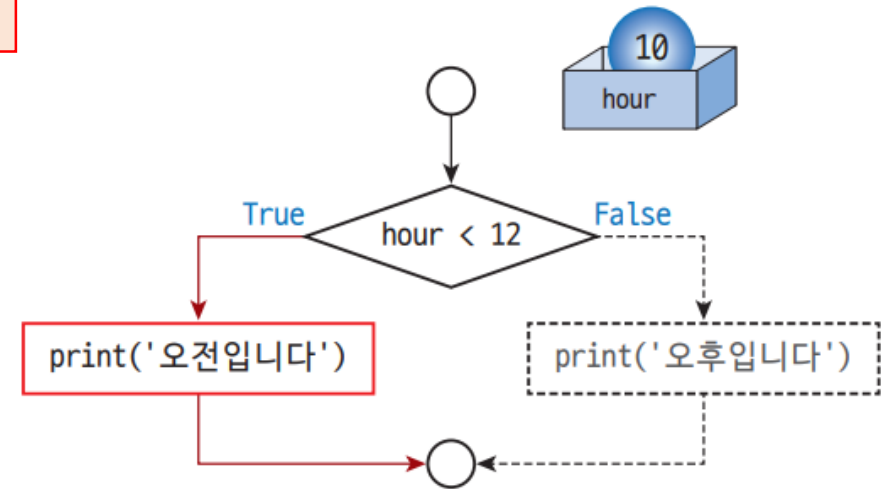
```
hour = 10
```

배타적인 관계

```
if hour < 12:  
    print('오전입니다.')  
else:  
    print('오후입니다.')
```

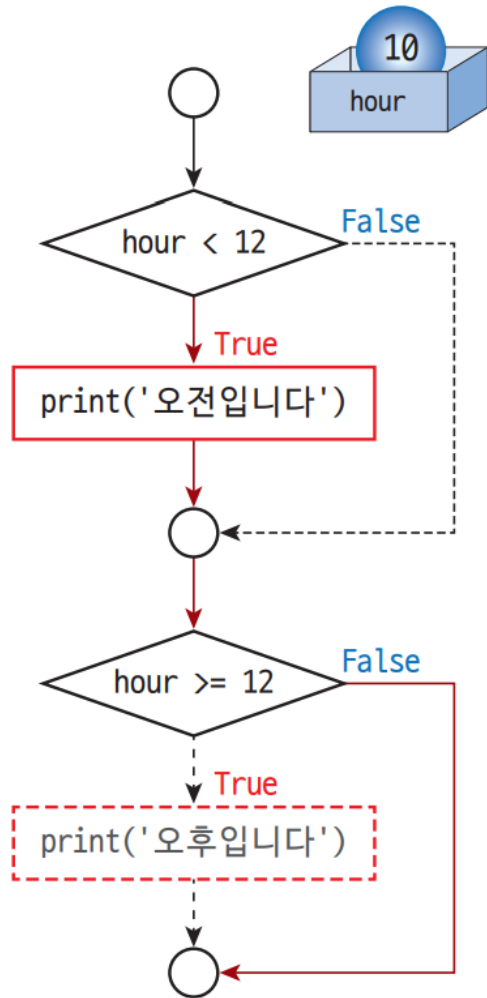
실행결과

오전입니다.

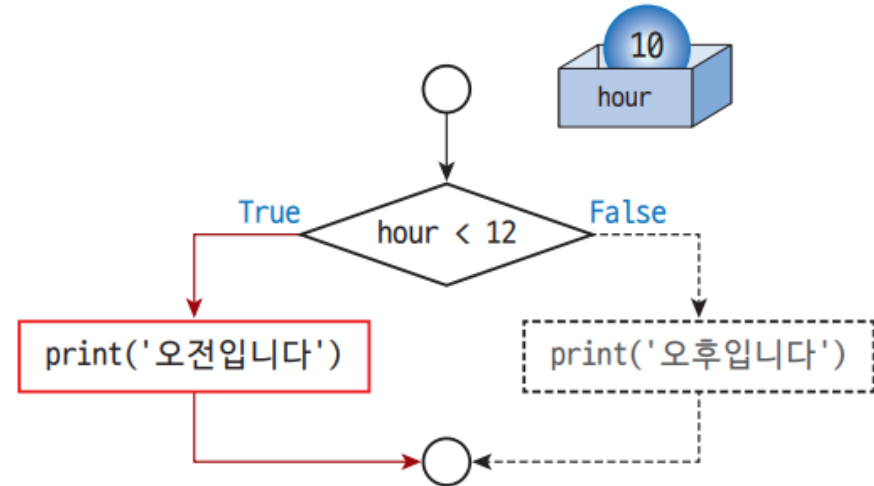


[그림 3-11] hour 값이 10일 때 if\_else\_hour\_test.py 코드의 흐름도

# 비교 : 어느쪽이 명확해 보이세요?



[그림 3-10] hour 값이 10일 때 코드 `if_hour_test.py`의 흐름도



[그림 3-11] hour 값이 10일 때 `if_else_hour_test.py` 코드의 흐름도

# 임의의 정수가 음수인지 아닌지 판단하기

코드 3-10 : if-else 문을 이용한 '음수' 혹은 '음수 아님'의 출력기능

if\_else\_minus\_test.py

```
num = -10
```

```
if num < 0:
```

```
    print(num, '은(는) 음수입니다.')
```

```
else:
```

```
    print(num, '은(는) 음수가 아닙니다.')
```

배타적인 관계

실행결과

-10 은(는) 음수입니다.



# if-else문으로 홀수/짝수를 판별 (배타적관계)

코드 3-11 : if-else 문을 이용한 '짝수' 혹은 '홀수'의 출력기능

if\_else\_even\_test.py

```
num = 10
```

```
if num % 2 == 0:
```

```
    print(num, '은(는) 짝수입니다.')
```

```
else:
```

```
    print(num, '은(는) 홀수입니다.')
```

배타적인 관계

실행결과

10 은(는) 짝수입니다.

# if문으로 구성된 블록 내 또 다른 if문이나 if-else 사용

코드 3-12 : 외부 if-else 문과 내부 if-else 문의 사용

if\_block.py

```
num = 100
if num < 0:
    print(num, '은(는) 음수입니다.')
else:
    print(num, '은(는) 음수가 아닙니다.')

# 짝수, 홀수는 음수가 아닐 때만 판별함
if num % 2 == 0:
    print(num, '은(는) 짝수입니다.')
else:
    print(num, '은(는) 홀수입니다.')
```

실행결과

100 은(는) 음수가 아닙니다.  
100 은(는) 짝수입니다.

- 바깥의 if-else문을 **외부 if-else문** 이라고 함
  - 외부의 if-else 조건문 블록을 살펴보면 if문은 변수 num의 값이 0보다 작을 때, 즉 음수일 때만 실행
  - else문은 num의 값이 음수가 아닐 때에만 실행
- else문 내의 if-else문을 **내부 if-else문** 이라고 함
  - if문은 변수 num을 2로 나누어 나머지가 없을 때, 즉 짝수일 때만 실행
  - 음수일 때는 else문 블록이 실행
- 변수 num의 값을 -100으로 바꾸면 다음과 같이 실행됨

num = -100으로 수정한 후의 실행 결과
-100 은(는) 음수입니다.



### LAB 3-3 : if 조건문의 응용

1. 게임 사용자의 게임점수(game\_score)을 입력받아서 1000점 이상이면 '고수입니다.'를 출력하고 1000점 미만이면 '입문자입니다.'를 출력하는 프로그램을 if-else 문을 이용하여 작성하시오.

게임점수를 입력하시오 : 800

```
game_score = 800
```

입문자입니다.

혹은

게임점수를 입력하시오 : 1300

```
game_score = 1300
```

고수입니다.

2. 두 정수를 입력받아 같을 경우 '두 값이 일치합니다.'를 출력하고 값이 다르면 '두 값이 일치하지 않습니다.'를 출력하는 프로그램을 if 문을 이용하여 작성하시오. 두 정수에 각각 100과 200이 할당되어 있는 경우와 300과 300이 할당되어 있는 경우의 출력문을 확인 하시오.

한 정수를 입력하시오 : 100  
다른 정수를 입력하시오 : 200  
두 값이 일치하지 않습니다.

혹은

한 정수를 입력하시오 : 300  
다른 정수를 입력하시오 : 300  
두 값이 일치합니다.

3. if 문의 복합 조건식을 이용해서 다음과 같은 기능을 수행하는 프로그램을 만들어보자.

1) 우선 '당신은 성인인가요(성인이면 1, 미성년이면 0)' 문을 통해서 성인인지 미성년인지를 구한다음 미성년이면(0이 입력되면) '당신은 미성년자입니다.'를 출력하고 프로그램을 종료한다.

당신은 성인인가요(성인이면 1, 미성년이면 0): 0

당신은 미성년자입니다.

2) 다음으로 성인이면(1이 입력되면) '결혼을 하셨나요(기혼이면 1, 미혼이면 0)' 질문을 통해서 기혼, 미혼을 입력으로 받아서 '당신은 결혼한 성인입니다.' 혹은 '당신은 결혼하지 않은 성인입니다.'를 다음과 같이 출력하도록 한다.

당신은 성인인가요(성인이면 1, 미성년이면 0): 1

결혼을 하셨나요(기혼이면 1, 미혼이면 0): 1

당신은 결혼한 성인입니다.

당신은 성인인가요(성인이면 1, 미성년이면 0): 1

결혼을 하셨나요(기혼이면 1, 미혼이면 0): 0

당신은 결혼하지 않은 성인입니다.

## 3.3.1 복합 조건식

- 더 정교한 조건을 걸어주기 위해 조건 연산자와 논리 연산자를 조합
- 모두 부울 값(True, False)을 반환한다는 공통점이 있음

비교연산 (<, <=, >=, >, !=, ==)

부울 연산 (True, False)

논리 연산 (and, or, not), in, not in, is, ...

- 비교 연산자는 연산자 왼쪽의 값과 오른쪽의 값이 해당 연산자의 조건을 만족할 시 True 아니면 False를 반환

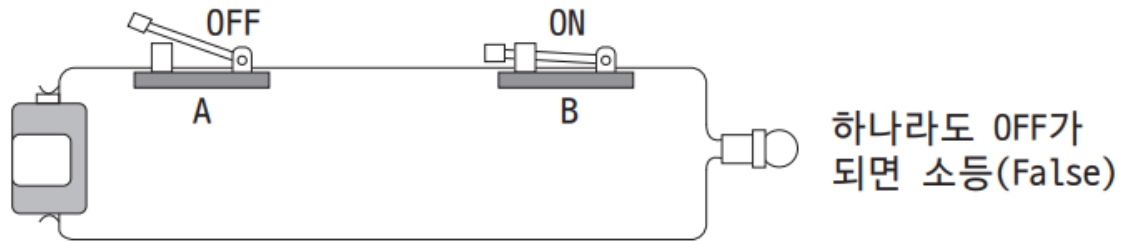
대화창 실습 : 조건식 실습

```
>>> 0 < 10          # 조건식이 참, 'True' 출력
True
>>> 4 > 10          # 조건식이 거짓, 'False' 출력
False
>>> 3 <= 10         # 조건식이 참, 'True' 출력
True
>>> 15 >= 10        # 조건식이 참, 'True' 출력
True
>>> 1 == 2          # 조건식이 거짓, 'False' 출력
False
>>> True or False   # 조건식이 참, 'True' 출력
True
>>> True and False  # 조건식이 거짓, 'False' 출력
False
```



# 논리 연산 and

- 입력 값 중에서 **False** 상태에 영향을 받는 특징이 있음



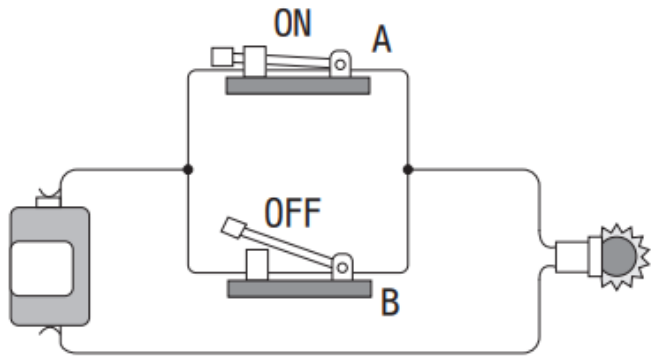
and

입력		출력
A	B	
True	True	True
True	False	False
False	True	False
False	False	False

[그림 3-13] and 연산을 수행하는 직렬 회로도와 논리 연산의 결과

# 논리 연산 or

- 출력 값이 입력 값의 True 상태에 영향을 받음



하나라도 ON이  
되면 점등(True)

or

입력		출력
A	B	
True	True	True
True	False	True
False	True	True
False	False	False

[그림 3-14] or 연산을 수행하는 병렬 회로도와 논리 연산의 결과

- 변수 a와 b에 저장된 값은 각각 10, 14이므로 두 조건문의 조건식이 모두 참(True)
- 실행 결과는 두 개의 print()문이 모두 실행됨

코드 3-13 : and와 or 조건문의 사용법

if\_and\_or\_test.py

```
a = 10
b = 14                                # 13으로 수정하면 첫 번째 조건문을 만족하지 않음
if (a % 2 == 0) and (b % 2 == 0):     # 첫 번째 조건문
    print('두 수 모두 짝수입니다.')
if (a % 2 == 0) or (b % 2 == 0):      # 두 번째 조건문
    print('두 수 중 하나 이상이 짝수입니다.')
```

실행결과

두 수 모두 짝수입니다.

두 수 중 하나 이상이 짝수입니다.

- ( $b = 13$ 으로 수정하면) 첫 번째 조건문의 조건식을 만족하지 못해 두 번째 조건문 내부의 print문만 실행됨

실행결과

두 수 중 하나 이상이 짝수입니다.



### LAB 3-4 : 복합 조건식의 이해

1. and 연산자를 사용하여 num 변수가 1과 10사이의 값을 가지면 True를 출력하는 부울식을 완성하여라.

```
>>> num = 2
```

```
>>> _____
```

```
True
```

2. and 연산자를 사용하여 age가 10보다 크고 19보다 작으면 '청소년입니다.'를 출력하는 부울식을 작성하여라. 그리고 age에 9와 12를 넣어서 그 결과를 다음과 같이 확인하여라.

```
age = 10
```

혹은

```
age = 12
```

```
청소년입니다.
```

## 3.3.2 복합 조건식으로 윤년 검사하기

- 윤년 **leap year**의 규칙

- 1) 연수가 4로 나누어 떨어지는 해는 윤년으로 한다(예를 들어 1992년, 2004년등)
- 2) 연수가 1의 조건에 만족함에도 100으로 나누어 떨어지는 해는 평년으로 한다 (예를 들어 1900년, 2100년, 2200년, 2300년 등)
- 3) 연수가 2의 조건에 만족함에도 400으로 나누어 떨어지는 해는 윤년으로 한다(예를 들어 2000년, 2400년등)

# 윤년 판별하기

코드 3-14 : 윤년을 판별하기 위한 코드

if\_leapyear\_test.py

```
# 윤년 판별하기
```

```
year = int(input('연도를 입력하세요 : '))
```

```
is_leap_year = ((year % 4 == 0) and (year % 100 != 0) or (year % 400 == 0))
```

```
print(year, '년은 윤년입니까?', is_leap_year)
```

실행결과

연도를 입력하세요 : 2000

2000 년은 윤년입니까? True

## 3.4 if-elif-else 문

- 많은 if문을 사용해 점수대별로 등급을 나누는 학점 산출기
  - 여러 개의 if문과 if문내의 and 조건을 적용하여 문제를 해결

점수	등급
100점 ~ 90점 이상	A
90점 미만 ~ 80점 이상	B
80점 미만 ~ 70점 이상	C
70점 미만 ~ 60점 이상	D
60점 미만	F



코드 3-15 : 'A','B','C','D','F' 등급 계산을 위한 if 문

if\_grade1.py

```
score = int(input('점수를 입력하세요 : '))  
if score >= 90 :                # 90 이상인 경우 'A'  
    grade = 'A'  
if score < 90 and score >= 80 : # 90 미만 80 이상인 경우 'B'  
    grade = 'B'  
if score < 80 and score >= 70 : # 80 미만 70 이상인 경우 'C'  
    grade = 'C'  
if score < 70 and score >= 60 : # 70 미만 60 이상인 경우 'D'  
    grade = 'D'  
if score < 60 :                # 60 미만인 경우 'F'  
    grade = 'F'  
print('당신의 등급은 :', grade)
```

실행결과

점수를 입력하세요 : 77  
당신의 등급은 : C

- 이전에 살펴본 간단한 if문보다 복잡하고 코드를 읽기가 어려웠음
- 세 번째 조건문에서 다음과 같은 잘못된 조건식이 들어가도 한 눈에 오류를 파악하기가 힘들

```
if score < 80 and score > 70 : # if score < 80 and score >= 70 : 의 오류 코드
```

- 각각의 if문의 의미를 하나하나 파악해야하기 때문에 오류의 가능성이 높아짐
- 이를 해결하기 위하여 다음과 같이 if-else 문을 적용

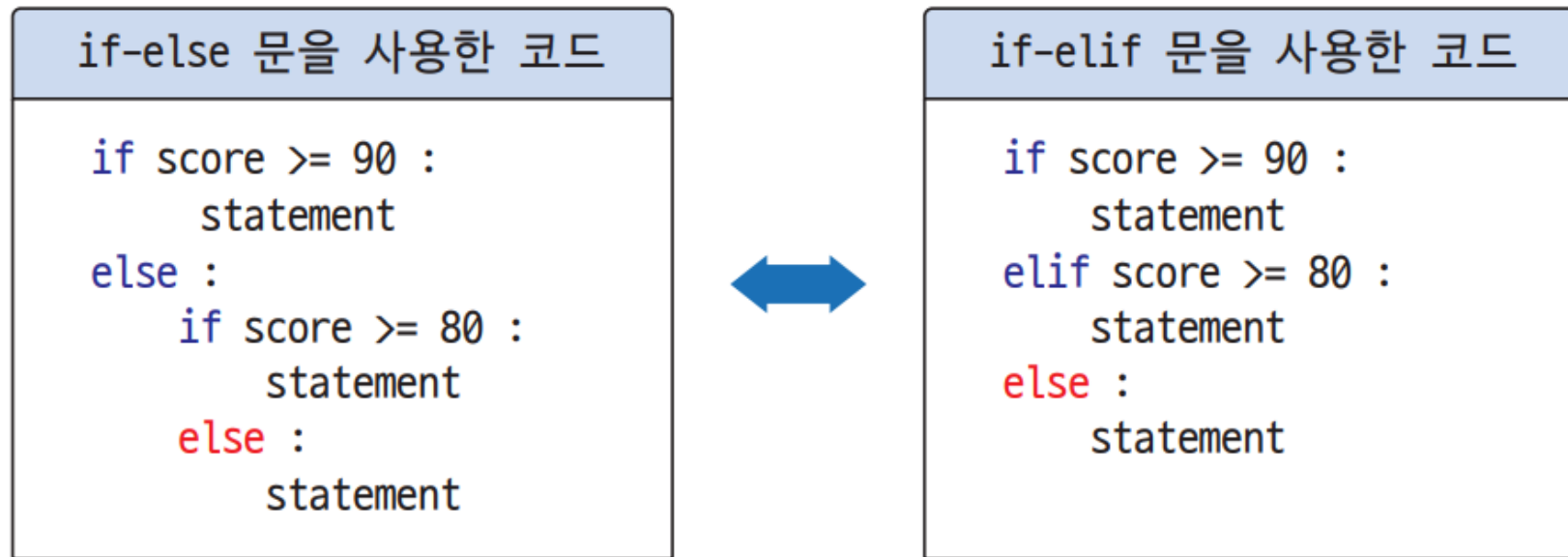
코드 3-16 : 'A','B','C','D','F' 등급 계산을 위한 복합 if 문

if\_grade2.py

```
score = int(input('점수를 입력하세요 : '))
if score >= 90:                      # 90 이상인 경우 'A'
    grade = 'A'
else:
    if score >= 80 :                  # 90 미만 80 이상인 경우 'B'
        grade = 'B'
    else:
        if score >= 70:              # 80 미만 70 이상인 경우 'C'
            grade = 'C'
        else:
            if score >= 60:           # 70 미만 60 이상인 경우 'D'
                grade = 'D'
            else:                     # 60 미만인 경우 'F'
                grade = 'F'
print('당신의 등급은 :', grade)
```

- 이전의 if문으로만 구성되어있던 [코드 3-15]보다는 읽기가 편해짐
- 오류의 가능성도 이전에 비해서 줄어듦
- if-else가 조건을 2개밖에 나타낼 수밖에 없기 때문에 가독성은 여전히 떨어짐
- 조건이 여러 개인 경우 if문에서 else문까지 가기 전에 조건을 더 걸어줄 수는 없을까? -> **elif문 사용하기**

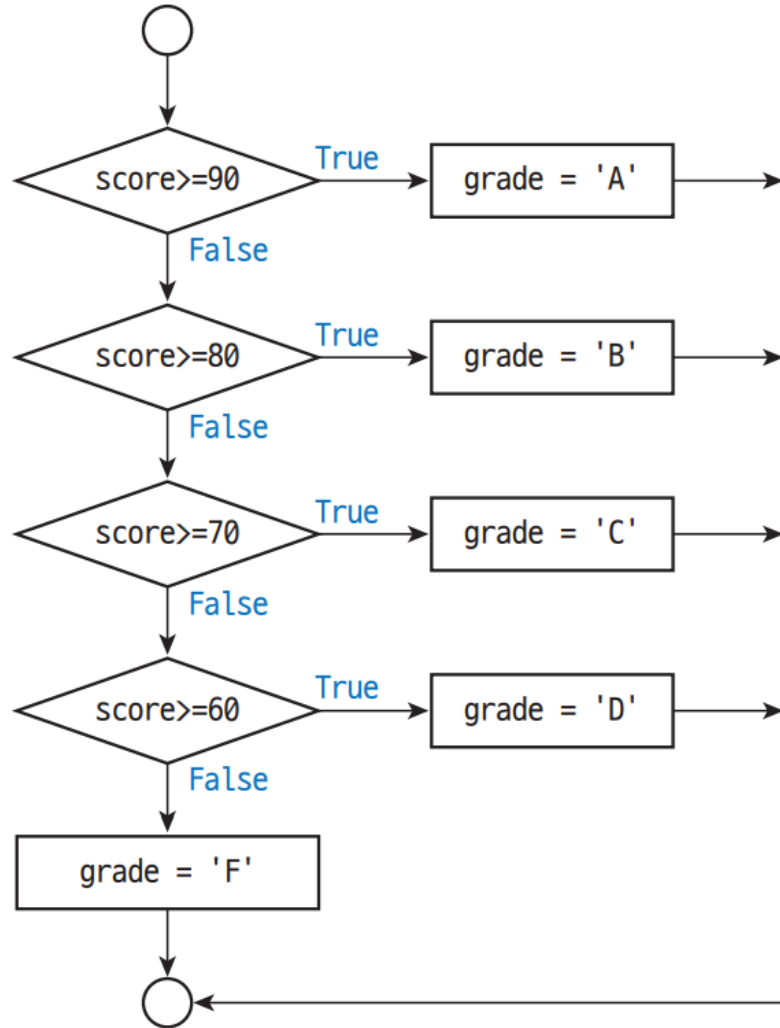
# if-else문과 elif문의 비교



[그림 3-15] 복합 if-else 문과 if-elif 문을 사용한 동일한 코드

- 왼쪽과 오른쪽의 코드는 동일
- 오른쪽의 코드가 들여쓰기도 더 적게하고 줄의 수도 더 줄어들어 코드를 이해하기가 더 편리해짐

# if-elif-else 문의 실행 흐름도



[그림 3-16] if-elif-else 문의 실행 흐름도

### 코드 3-17 : if-elif-else 문으로 구성된 등급계산기

if\_elif\_grade.py

```
score = int(input('점수를 입력하세요 : '))  
if score >= 90:      # 90 이상인 경우 'A'  
    grade = 'A'  
elif score >= 80:    # 'A'가 아닌 경우, 80 이상이면 'B'  
    grade = 'B'  
elif score >= 70:    # 'B'도 아닌 경우, 70 이상이면 'C'  
    grade = 'C'  
elif score >= 60:    # 'C'도 아닌 경우, 60 이상이면 'D'  
    grade = 'D'  
else:                # 그 외의 경우 'F'  
    grade = 'F'  
print('당신의 등급은 :', grade)
```

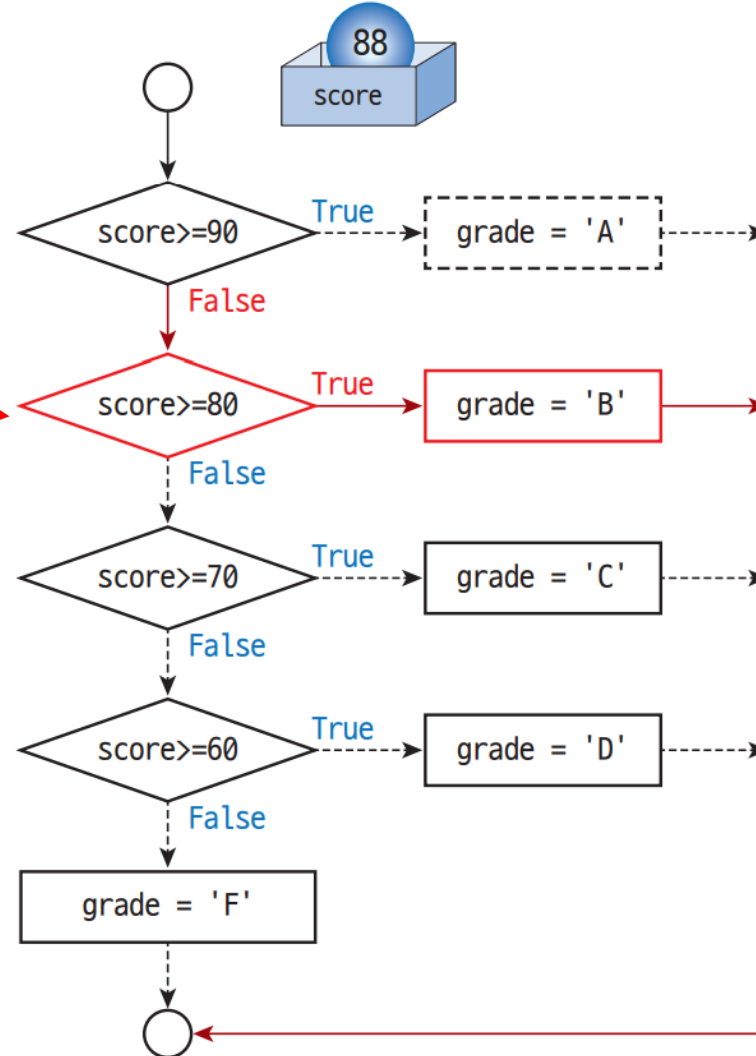
#### 실행결과

점수를 입력하세요 : 88

당신의 등급은 : B

# if-elif-else문의 구조

score 변수의 값이 88이면 다음과 같은 실행 흐름을 가지게 된다



[그림 3-17] score 값이 88일 때 if-elif-else 문의 수행 흐름





### LAB 3-5 : if-elif-else 문을 사용한 다중 조건식

1. 사용자로부터 자동차의 속도(speed)를 km/h 단위의 정수로 입력받도록 하자. 자동차의 속도가 100km/h 이상이면 '고속', 100km/h 미만 60km/h 이상이면 '중속', 60km/h 미만이면 '저속'을 출력하는 프로그램을 if-elif-else 문을 이용하여 작성하여라.

자동차의 속도를 입력하세요(단위 : km/h ): 13

저속

혹은

자동차의 속도를 입력하세요(단위 : km/h ): 130

고속

## 3.5 for 반복문

- 특정한 작업을 여러 번 되풀이해서 수행하고 싶을 경우 사용
- 반복문에는 for 문과 while 문 두 종류가 있음
- for 문은 반복의 횟수가 미리 정해져 있는 경우, while 문은 반복 횟수는 알지 못하지만 반복하는 조건이 명확한 경우에 사용

- 반복문을 사용하지 않고 5번 반복해 출력하려면 다음과 같이 print() 함수를 5번에 걸쳐 사용
- 1000회 반복시 매우 비효율적임

코드 3-18 : print() 함수의 호출을 통한 반복적 수행

print\_welcome.py

```
print('Welcome to everyone!!')  
print('Welcome to everyone!!')  
print('Welcome to everyone!!')  
print('Welcome to everyone!!')  
print('Welcome to everyone!!')
```

실행결과

```
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!
```

- range() 함수는 특정한 구간의 정수 **열sequence**을 반복해서 생성함
- for 문에서 순환을 위한 용도이다

```
for i in range( n ):
```



새로운 변수



반복 실행 횟수

횟수가 정해져있거나 1씩 증가하는 숫자를 써야할 때 사용  
i는 0부터 n-1까지 증가함

[그림 3-18] for in range() 구문의 사용법

코드 3-19 : for 문을 이용한 반복적 수행

print\_welcome\_with\_for1.py

```
for i in range(5):  
    print('Welcome to everyone!!')
```

실행결과

```
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!
```

10번 반복시 range() 괄호 내의 값만 10으로 고쳐주면 됨

```
for i in range(10):  
    print('Welcome to everyone!!')
```



#### NOTE : 루프 제어변수의 익명화

위 반복문에서 새롭게 할당되는 변수 `i`는 실행문에서 사용되지 않는 루프변수이므로 다음과 같이 언더스코어(`_`)를 대신 넣어서 익명화 시킬 수 있다.

```
for _ in range(10):  
    print('Welcome to everyone!!!!')
```

코드 3-20 : for 문을 이용한 반복적 수행 - 10회 수행

print\_welcome\_with\_for2.py

```
for i in range(10):  
    print(i, 'Welcome to everyone!!')
```

#### 실행결과

```
0 Welcome to everyone!!  
1 Welcome to everyone!!  
2 Welcome to everyone!!  
3 Welcome to everyone!!  
4 Welcome to everyone!!  
5 Welcome to everyone!!  
6 Welcome to everyone!!  
7 Welcome to everyone!!  
8 Welcome to everyone!!  
9 Welcome to everyone!!
```

- 반복문에서 사용되는 변수는 i, j, k, l,... 과 같은 알파벳 문자를 할당
- 이러한 변수를 C나 Java에서는 루프(loop) 제어변수 라고 함



### LAB 3-6 : 반복문을 이용해서 다음 코드를 작성해 보자

1. for 반복문에서 언더스코어 루프 제어변수를 사용하여 다음과 같이 Hello, Python!을 5번 출력하는 프로그램을 작성해 보자.

```
Hello, Python!  
Hello, Python!  
Hello, Python!  
Hello, Python!  
Hello, Python!
```

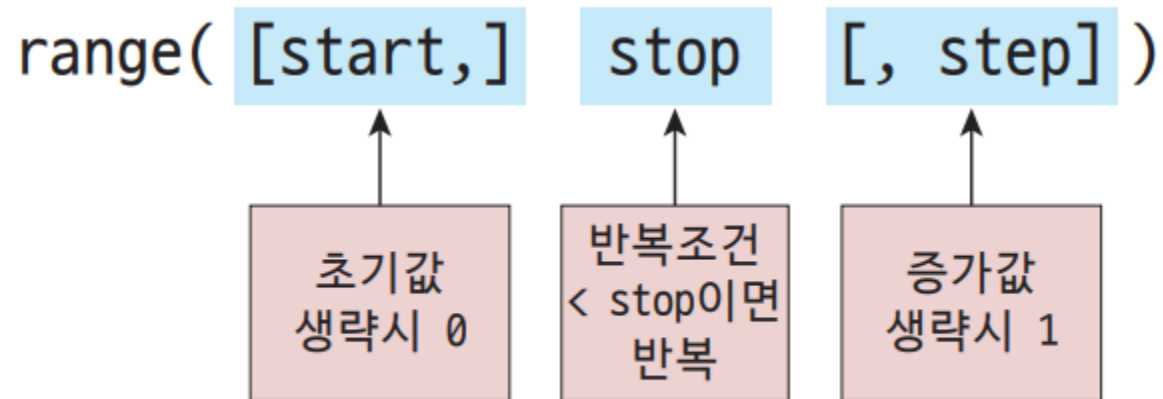
2. i라는 루프변수를 사용해서 0에서 4까지의 정수를 출력하는 프로그램을 작성하시오.

```
0  
1  
2  
3  
4
```



# range() 함수의 사용법

- range(0, 5)와 같이 주어진 시작 값에서 마지막 값 사이의 연속적인 정수들을 생성할 수도 있으며, range(0, 5, 2)와 같이 마지막에 증가치 값을 넣어 줄 수도 있다.
- range(0, 5, 1)과 같이 호출할 경우 마지막의 1은 디폴트 간격(step) 값으로 1씩 더하면서 값을 변경하라는 의미



[그림 3-19] range() 함수의 사용법 : [start]와 [step]값은 생략할 수 있다.



#### NOTE : range() 함수의 인자

range() 함수는 실수 값을 인자로 가질 수 없다. 반면 나중에 배우게 될 Numpy 모듈의 arange()라는 함수는 실수의 시작 값, 종료 값, 스텝 값을 가질 수 있다.

## 대화창 실습 : range() 함수의 활용과 리스트

```
>>> list(range(5))           # 0에서 4사이의 정수열을 생성
[0, 1, 2, 3, 4]

>>> list(range(0, 5))        # list(range(5))와 동일한 결과
[0, 1, 2, 3, 4]

>>> list(range(0, 5, 1))      # list(range(0, 5))와 동일한 결과
[0, 1, 2, 3, 4]

>>> list(range(0, 5, 2))      # 생성하는 값을 2씩 증가시킴
[0, 2, 4]

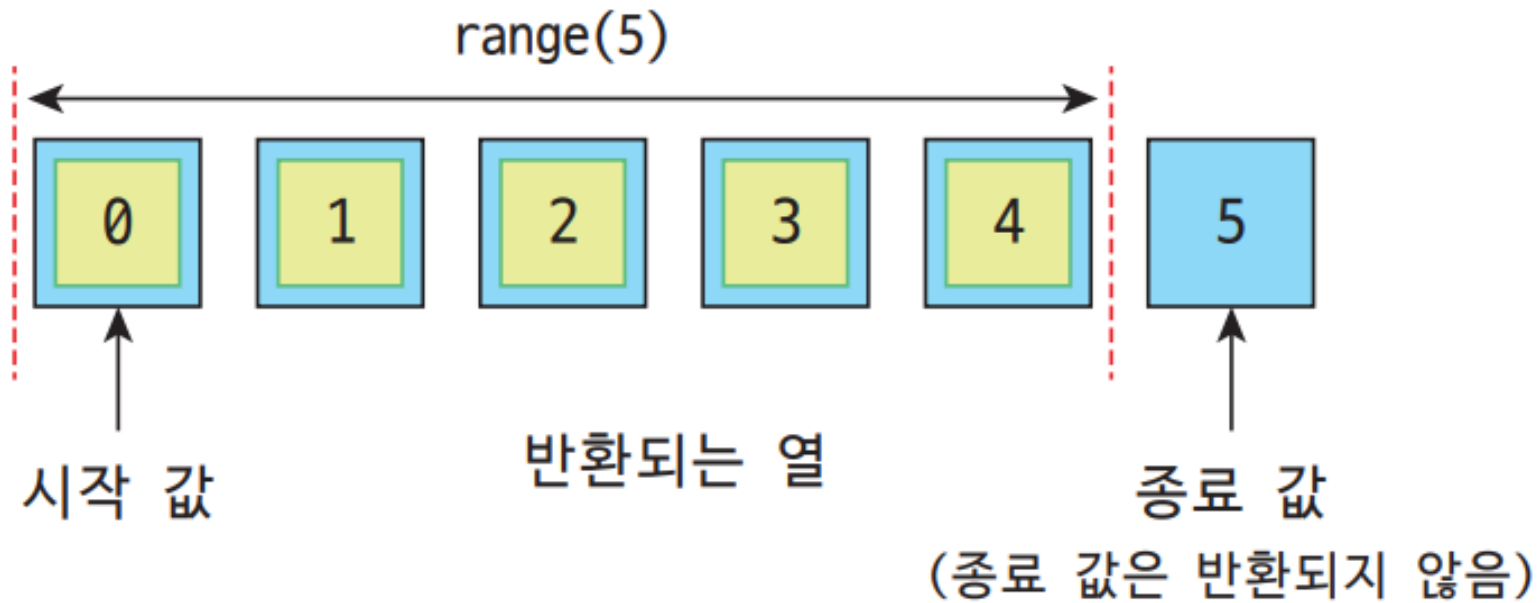
>>> list(range(2, 5))         # 2에서 5-1까지의 연속된 수 2, 3, 4을 생성
[2, 3, 4]

>>> list(range(0, 10, 2))     # 0에서 9사이의 짝수 리스트 생성
[0, 2, 4, 6, 8]

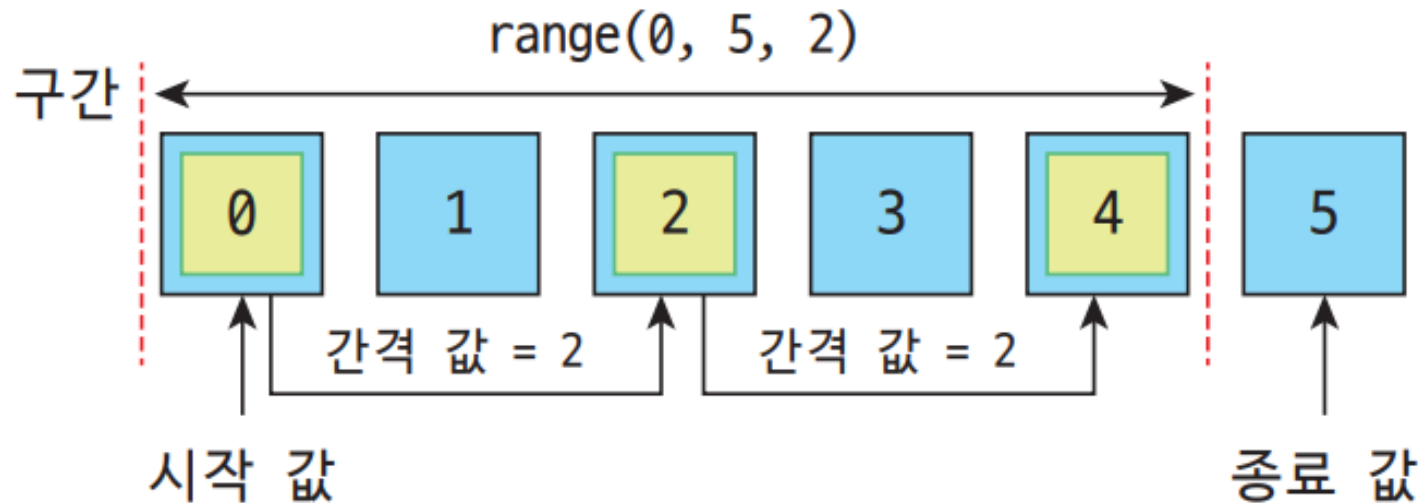
>>> list(range(1, 10, 2))     # 0에서 9사이의 홀수 리스트 생성
[1, 3, 5, 7, 9]

>>> list(range(-2, -10, -2))  # 음수 간격 값을 이용하여 -2,-4,-6,-8을 생성
[-2, -4, -6, -8]
```

- range(5)의 시작 값과 종료 값, 그리고 반환되는 열



- `range(2, 5)`는 2이상 5미만의 정수열 `[2, 3, 4]`를 생성하며,  
`range(0, 5, 2)`는 0에서 5미만의 정수들을 생성할 때 2씩 증가시키므로 `[0, 2, 4]`의 정수를 생성
- 양수 간격 값을 사용할 때는 `range(0, 5, 1)`과 같이 반드시 시작 값이 종료 값보다 작아야 하며, 음수 간격 값을 사용할 때는 `range(-2, -10, -2)`와 같이 시작 값이 반드시 종료 값보다 커야 한다





### LAB 3-7 : range() 함수의 응용

1. range() 함수와 list() 함수를 사용하여 1이상 100이하의 자연수 리스트를 다음과 같이 만드시오.

```
[1, 2, 3, 4, 5, (생략), 100]
```

2. 1 이상 100 이하의 짝수 리스트를 만드시오.
3. 1 이상 100 이하의 홀수 리스트를 만드시오.
4. -100보다 크고 0보다 작은 음수 리스트를 만드시오.

코드 3-21 : range() 함수를 이용한 for 문의 제어

for\_in\_range\_test1.py

# for in range 표현식

for i in range(5):

print(i)

실행결과

0

1

2

3

4

- print(i)에서 수를 출력한 후 매번 줄 바꿈을 하므로 결과를 보는 것이 불편함

- 아래와 같은 방법을 사용하여 디폴트 문자를 공백문자로 변경

```
print(i, end = ' ')
```

코드 3-22 : for 문의 제어와 print() 함수의 end 키워드 인자 사용방법

for\_in\_range\_test2.py

```
# for in range 표현식 1
for i in range(5):
    print(i, end = ' ')
# for in range 표현식 2
for i in range(0, 5):
    print(i, end = ' ')
```

실행결과

0 1 2 3 4 0 1 2 3 4



### 코드 3-23 : for 문의 제어와 print() 함수의 end 키워드 인자 사용방법

for\_in\_range\_test3.py

# for in range 표현식 1

```
for i in range(5):  
    print(i, end = ' ')  
print()
```

# for in range 표현식 2

```
for i in range(0, 5):  
    print(i, end = ' ')  
print()
```

실행결과

0 1 2 3 4

0 1 2 3 4

코드 3-24 : range() 함수를 이용한 for 문의 제어와 간격값 사용법

for\_in\_range\_test4.py

# 표현식 1 : 2에서 5-1까지 연속값 2, 3, 4 출력

```
for i in range(2, 5):
```

```
    print(i, end = ' ')
```

```
print()
```

# 표현식 2 : 간격 값을 사용하여 0, 2, 4, 6, 8 출력

```
for i in range(0, 10, 2):
```

```
    print(i, end = ' ')
```

```
print()
```

# 표현식 3 : 음수 간격 값 사용, -2, -4, -6, -8 출력

```
for i in range(-2, -10, -2):
```

```
    print(i, end = ' ')
```

실행결과

2 3 4

0 2 4 6 8

-2 -4 -6 -8

## 3.5.1 반복문의 활용

- 1에서 10까지의 정수의 합 구하기

코드 3-25 : 연속적인 값의 생성과 누적 덧셈

for\_sum\_ex1.py

```
s = 0
```

```
for i in range(1, 11):
```

```
    s = s + i
```

```
print('1에서 10까지의 합:', s)
```

실행결과

1에서 10까지의 합: 55



### 주의 : `sum()` 함수와 이름의 중복사용

위의 코드에서 `sum`이라는 변수를 사용하게 되면 파이썬 내장함수 `sum()`과 헷갈릴 수 있다. 동일한 모듈내에 `sum = 00`이라는 변수와 `sum(numbers)`라는 함수를 호출하게 되면 오류가 발생한다. 따라서 `sum()` 함수와 중복되지 않도록 변수의 이름을 `s`나 `total`로 사용할 것을 권장한다.



## NOTE : 반복문에서 초기화의 중요성, 누적 연산

[코드 3-25]의 프로그램에서 for 문이 반복되면 다음과 같이  $i$  값과  $s$  값이 변하게 된다.  $i$ 는 매 반복 루프가 돌 때마다 그 값이 변하며 이 때문에  $s$  값도 증가하게 된다.

$i$ 는 매번 1씩 증가하여 1, 2, 3, ...이 된다. 그리고 초기상태의  $s = 0$ 이므로 첫 번째 반복문 수행시  $0+1$ 이  $s$ 에 할당되지만, 매번  $s + i$  연산을 수행하기 때문에 아래와 같이  $0+1$ ,  $0+1+2$ ,  $0+1+2+3$ , ... 연산을 수행하게 된다.

마침내 10번째 반복이 수행되면 최종적으로  $s$ 는 55 값을 가지며 반복은 중단된다.

반복	$i$ 값	$s$ 값	반복여부	$s$ 의 계산
1번째	1	1	반복	$0+1$
2번째	2	3	반복	$0+1+2$
3번째	3	6	반복	$0+1+2+3$
4번째	4	10	반복	$0+1+2+3+4$
5번째	5	15	반복	$0+1+2+3+4+5$
6번째	6	21	반복	$0+1+2+3+4+5+6$
7번째	7	28	반복	$0+1+2+3+4+5+6+7$
8번째	8	36	반복	$0+1+2+3+4+5+6+7+8$
9번째	9	45	반복	$0+1+2+3+4+5+6+7+8+9$
10번째	10	55	반복 중단	$0+1+2+3+4+5+6+7+8+9+10$

이전 연산의 결과가 누적되어

이전 연산의 결과가 누적되어

이전 연산의 결과가 누적되어  
더해지게 된다

...

### 코드 3-26 : 누적 덧셈의 중간 결과 출력하기

for\_sum\_ex2.py

```
s = 0
for i in range(1, 11):
    s = s + i
    print('i = {}, s = {}'.format(i, s) )

print('1에서 10까지의 합:', s)
```

### 실행결과

```
i = 1, s = 1
i = 2, s = 3
i = 3, s = 6
i = 4, s = 10
i = 5, s = 15
i = 6, s = 21
i = 7, s = 28
i = 8, s = 36
i = 9, s = 45
i = 10, s = 55
1에서 10까지의 합: 55
```

- 코드 for\_sum\_ex1.py에 비해  
전체 수행과정을 살펴볼 수 있어 이해하기 편하다.



### LAB 3-8 : 누적 덧셈의 응용

1. 앞서 배운 누적 덧셈을 응용하여 1에서 100까지 정수의 합을 구하여 출력하여라(힌트 : range()의 구간이 1에서 100까지가 되도록 하자).
2. range() 함수의 step 값을 이용하여 1에서 100까지 정수 중에서 짝수의 합을 구하여 출력하여라(힌트 : range() 함수의 시작 값은 0으로 하고 step 값을 2로 하여라).
3. range() 함수의 step 값을 이용하여 1에서 100까지 정수 중에서 홀수의 합을 구하여 출력하여라(힌트 : range() 함수의 시작 값은 1로 하고 step 값을 2로 하여라).

# 임의의 양의 정수를 입력받아 1부터 n까지의 정수 출력

코드 3-27 : 사용자로부터 입력을 받은 후 누적 합계 값 구하기

for\_sum\_input1.py

```
n = int(input('합계를 구할 수를 입력하세요 : '))
```

```
s = 0
```

```
for i in range(0, n) :
```

```
    s = s + (i+1)
```

```
print('1부터 {}까지의 합은 {}'.format(n, s))
```

- 여러 가지 방식 중에서 가장 **이해하기 쉽고** 다른 사람이 **읽기 좋은 코딩방식**을 사용

실행결과

합계를 구할 수를 입력하세요 : 100

1부터 100까지의 합은 5050



- range() 함수의 시작 값을 1로 하고 마지막 값을  $n+1$ 로 하여 1부터  $n$ 까지의 정수를 더하기

코드 3-28 : 사용자로부터 입력을 받은 후 누적 합계 값 구하기

for\_sum\_input2.py

```
n = int(input('합계를 구할 수를 입력하세요 : '))
s = 0

for i in range(1, n+1) :
    s = s + i
print('1부터 {}까지의 합은 {}'.format(n, s))
```

실행결과

합계를 구할 수를 입력하세요 : 100

1부터 100까지의 합은 5050

## 3.5.2 팩토리얼factorial 구하기

- $n$ 이 임의의 자연수 일때 1에서  $n$ 까지의 모든 자연수를 곱한 값이다.

$$n! = \prod_{k=1}^n k = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$$

코드 3-29 : for 반복문을 이용한 5 팩토리얼(5!) 계산

for\_factorial.py

```
n = int(input('수를 입력하세요 : '))
fact = 1
for i in range(1, n+1):
    fact = fact * i

print('{}! = {}'.format(n, fact))
```

- 변수 fact의 초기 값은 1로 두고 덧셈(+) 연산 대신 곱셈(\*) 연산을 for문 안에서 사용

실행결과

수를 입력하세요 : 5

5! = 120



#### NOTE : 파이썬과 정수 표현의 한계

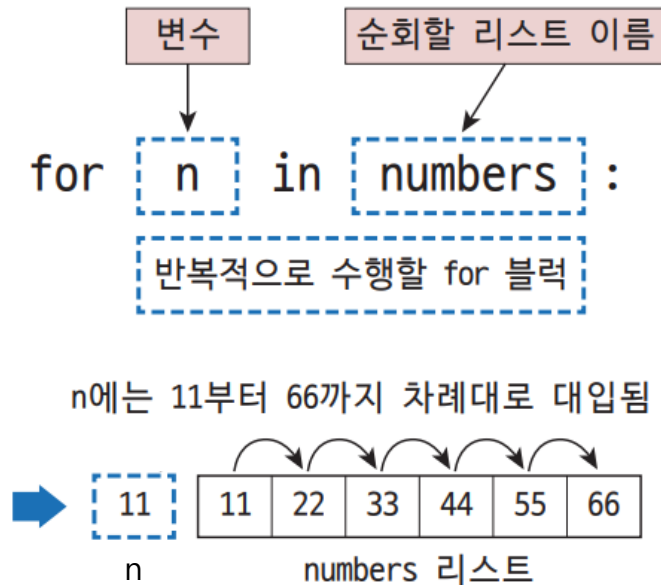
팩토리얼을 구하는 [코드 3-29]를 수정하여 n 값을 50으로 바꾼다면 다음과 같은 엄청나게 큰 수를 출력한다.

$50! = 30414093201713378043612608166064768844377641568960512000000000000.$

C나 Java와 같은 프로그래밍 언어는 정수의 크기가 4 바이트 형으로 정해져 있어서 일정한 크기 이상의 정수는 표현할 수 없다. 예를 들어 Java의 long형 정수의 범위는  $-9223372036854775808 \sim 9223372036854775807$  까지이다. 이와 달리 파이썬은 정수 표현의 한계가 없다. 이 점이 파이썬의 또 다른 큰 장점이다.

## 3.5.3 for 문과 리스트

- for in 구문
  - 반복문 키워드 for 와 in 사이에 계속 새롭게 할당할 변수 n을 선언
  - in 뒤에 리스트 자료형을 넣어 리스트를 차례대로 순회하는 실행이 가능



[그림 3-22] for - in 구문에서 적용되는 numbers 리스트의 순회 방문 원리

코드 3-30 : for 문을 이용한 리스트의 정수 객체 순회

for\_in\_numbers.py

```
numbers = [11, 22, 33, 44, 55, 66]
```

```
for n in numbers:  
    print(n, end = ' ')
```

실행결과

11 22 33 44 55 66

코드 3-31 : for 문을 이용한 리스트의 실수 객체 순회

for\_in\_f\_numbers.py

```
f_numbers = [1.1, 2.5, 3.7, 5.6, 9.2, 11.3, 6.8]
```

```
for f in f_numbers:  
    print(f, end = ' ')
```

실행결과

1.1 2.5 3.7 5.6 9.2 11.3 6.8

- for in 구문의 in 다음에 범위를 지정하는 함수 range()가 아닌 numbers 라는 리스트가 있음
- 실수 리스트도 for in문을 통해 순회가 가능

- for문을 이용하며 문자열을 원소로 가지는 리스트의 원소들을 출력

코드 3-32 : for 문을 이용한 리스트의 문자열 객체 순회

for\_in\_str\_list.py

```
summer_fruits = ['수박', '참외', '체리', '포도']
```

```
for fruit in summer_fruits:  
    print(fruit, end = ' ')
```

실행결과

수박 참외 체리 포도

- 누적 덧셈의 기능을 활용하여 리스트 내에 있는 정수 항목 값들의 합을 구하는 프로그램

코드 3-33 : 리스트 항목내 정수 값들의 누적 덧셈

for\_sum1.py

```
numbers = [10, 20, 30, 40, 50]
s = 0
for n in numbers:
    s = s + n
print('리스트 항목 값의 합 :', s)
```

실행결과

리스트 항목 값의 합 : 150



- 리스트 원소들의 합은 for 문을 사용하지 않고 내장함수 sum()을 사용하여 간편하게 합계를 구하는 것도 가능

코드 3-34 : sum() 함수의 사용

for\_sum2.py

```
numbers = [10, 20, 30, 40, 50]
```

```
print('리스트 항목 값의 합 :', sum(numbers))
```

실행결과

리스트 항목 값의 합 : 150

대화창 실습 : 대화형 모드를 통한 1에서 100까지의 합

```
>>> print('1에서 100까지의 합 :', sum(range(1, 101)))
```

```
1에서 100까지의 합 : 5050
```

- 문자열 자료형은 list() 함수를 이용하여 리스트 객체로 만드는 것이 가능

대화창 실습 : 대화형 모드를 통한 str 자료형의 리스트화

```
>>> st = 'Hello'
```

```
>>> list(st)
```

```
['H', 'e', 'l', 'l', 'o']
```

- 문자열 'Hello'를 list() 함수로 형 변환시켜 문자 원소들을 리스트로 만드는 것이 가능
- for in 구문 뒤에 문자열을 위치시켜 문자 단위로 분리해 순회할 수 있다.

코드 3-35 : for 반복문에서 문자열의 사용

for\_in\_hello.py

```
for ch in 'Hello':  
    print(ch, end = ' ')
```

실행결과

H e l l o

## 3.6 중첩 for 루프

- 이중 for문 **nested for loop**
  - for문 안에 for문을 다시 넣음
- 구구단의 구조에 대해 살펴보면, 2~9단까지 있으며 1에서 9를 단마다 곱하여 화면에 출력함
- 이를 구현하기 위해서는 for 문 안에 for 문을 다시 넣는 **이중 for문**이 필요

### 코드 3-36 : 중첩 for 문을 사용한 구구단 출력

double\_for.py

```
for i in range(2, 10):          # 외부 for 루프
    for j in range(1, 10):      # 내부 for 루프
        print('{}*{} = {:2d}'.format(i, j, i*j), end = ' ')
    print()                    # 내부 루프 수행 후 줄바꿈을 함
```

#### 실행결과

```
2*1= 2 2*2= 4 2*3= 6 2*4= 8 2*5=10 2*6=12 2*7=14 2*8=16 2*9=18
3*1= 3 3*2= 6 3*3= 9 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27
4*1= 4 4*2= 8 4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36
5*1= 5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45
6*1= 6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54
7*1= 7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63
8*1= 8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72
9*1= 9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

- double\_for.py의 이중 for 루프는 내부 루프와 외부 루프를 가짐

```
외부
[ 내부
  for i in range(2,10):
    for j in range(1,10):
      print('{} * {} = {}'.format(i, j, i*j))
```

- 루프의 실행구조를 표로 나타내어 보기

i = 2 일 때	i = 3일 때	i = 4일 때	...	i = 9일 때
j = 1 : 2 * 1	j = 1 : 3 * 1	j = 1 : 4 * 1	...	j = 1 : 9 * 1
j = 2 : 2 * 2	j = 2 : 3 * 2	j = 2 : 4 * 2	...	j = 2 : 9 * 2
j = 3 : 2 * 3	j = 3 : 3 * 3	j = 3 : 4 * 3	...	j = 3 : 9 * 3
j = 4 : 2 * 4	j = 4 : 3 * 4	j = 4 : 4 * 4	...	j = 4 : 9 * 4
j = 5 : 2 * 5	j = 5 : 3 * 5	j = 5 : 4 * 5	...	j = 5 : 9 * 5
j = 6 : 2 * 6	j = 6 : 3 * 6	j = 6 : 4 * 6	...	j = 6 : 9 * 6
j = 7 : 2 * 7	j = 7 : 3 * 7	j = 7 : 4 * 7	...	j = 7 : 9 * 7
j = 8 : 2 * 8	j = 8 : 3 * 8	j = 8 : 4 * 8	...	j = 8 : 9 * 8
j = 9 : 2 * 9	j = 9 : 3 * 9	j = 9 : 4 * 9	...	j = 9 : 9 * 9

- 이중 for 루프나 삼중 for 루프의 경우 코드를 이해하는 것이 어려워지기 때문에 중첩 루프는 삼중 루프 이상의 구조를 잘 사용하지 않음



## 3.6.1 중첩 for 루프를 이용한 패턴 만들기

- 다음과 같은 패턴 만들어 보기

```
#  
 #  
  #  
   #  
    #  
     #  
      #
```

### 코드 3-37 : 이중 for 루프를 사용한 패턴 생성하기

#### double\_for\_pattern.py

```
n = 7
# 외부 for 루프는 n번 수행, i는 0,1,2,3,4,5,6 까지 증가
for i in range(n):
    st = ''
    for j in range(i): # 내부 for 루프는 i번 수행
        st = st + ' ' # 공백을 i개 추가함
    print(st + '#') # 공백 추가 후 '#'출력
```

- 이중 for 루프를 사용하여 외부 루프에서는 i 값이 0에서 1씩 증가하도록 하고 내부 루프에서는 i의 갯수만큼의 공백을 '#' 표시 앞에 추가

### 코드 3-38 : for 루프와 \*를 사용한 패턴 생성하기

#### for\_pattern.py

```
n = 7
# 외부 for 루프는 n번 수행, i는 0,1,2,3,4,5,6 까지 증가
for i in range(n):
    print(' ' * i + '#') # 공백을 i번 추가한 후 '#'출력
```

- 이중 for 루프를 사용하지 않고 다음과 같이 print문 내에 공백의 출력횟수를 지정하는 방식 (' ' \* i)
- for\_pattern.py의 방식이 더 이해하기 쉽고 간단함



### LAB 3-9 : 패턴 출력 응용

1. 이중 for 루프를 이용하여 다음과 같은 패턴을 출력하여라.

```
      #  
     #  
    #  
   #  
  #  
 #  
#
```

## 3.6.2 상향식 문제풀이 기법

- 이중 for 루프를 사용하여 다음과 같은 패턴 만들어보기

```
+  
+++  
+++++  
+++++++  
+++++++++
```

# 4, 3, 2, 1, 0을 순서대로 출력하는 프로그램

코드 3-39 : 4, 3, 2, 1, 0을 순서대로 출력하는 프로그램

pattern\_test1.py

```
n = 5
for i in range(n):          # i는 0,1,2,3,4 까지 증가
    print(n - (i + 1), end = ' ') # n이 5이므로 n-(i+1)은 4,3,2,1,0이 됨
```

실행결과

4 3 2 1 0

# '+' 문자의 증가 패턴인 1, 3, 5, 7, 9를 출력

코드 3-40 : 1, 3, 5, 7, 9를 순서대로 출력하는 프로그램

pattern\_test2.py

```
n = 5
for i in range(n):           # i는 0,1,2,3,4 까지 증가
    print(2 * i + 1, end = ' ') # 2 * i + 1은 1,3,5,7,9가 됨
```

실행결과

1 3 5 7 9

- 앞서 완성한 두 가지 기능을 바탕으로 삼각형 패턴을 출력해보기
- 패턴을 출력하기 위해 우선 공백을 `for j in range(n - i - 1):`를 이용하여 출력한 다음, '+' 패턴을 `for j in range(2 * i + 1):`를 이용하여 출력
- 이와 같은 기법을 **상향식** `Bottom up` 문제풀이 기법이라고 함
  - 작은 기능을 구현하여 이 기능을 바탕으로 전체 기능을 구현하는 것

# 삼각형 패턴을 출력하는 기능

코드 3-41 : 삼각형 패턴을 출력하는 기능

triangle\_pattern1.py

```
n = 5
for i in range(n):
    for j in range(n - (i + 1)):          # 공백을 출력함
        print(' ', end = '')
    for j in range(2 * i + 1):          # '+'를 출력함
        print('+', end = '')
    print()
```





NOTE : Think big but start small.

문제 해결을 위한 프로그래밍 작성시에는 항상 `pattern_test1.py`, `pattern_test2.py`와 같은 작은 기능을 수행하는 프로그램을 만들어 보고 이 기능이 제대로 잘 동작하는가를 충분히 테스트하여야 한다. 작은 기능조차 제대로 동작하지 않는다면 큰 문제는 절대로 해결되지 않는다.

처음에 프로그램을 시작하는 개발자들은 자신이 만들 목표만 크게(BIG) 생각하고 작은 기능들을 어떻게 구현해야 하는가를 모르는 경우가 많다. 이와 같이 하지 말고 소규모 코딩으로 시작해서 자신이 구현하고자 하는 기능이 타당성이 있는가를 검토하는 개발 방법이 바람직하다.

# 삼각형 패턴을 출력하는 기능을 가진 짧은 코드

코드 3-42 : 삼각형 패턴을 출력하는 기능을 가진 짧은 코드

triangle\_pattern2.py

```
n = 5
for i in range(n):
    print(' ' * (n - (i + 1)), end = '')
    print('+ ' * (2 * i + 1))
```

## 3.6.3 소수 구하기

- 이중 for문을 활용해 소수를 구하는 프로그램을 구현
- 소수란 1과 자기 자신 이외의 약수를 가지지 않는 수
- 다음과 같은 코드로 구현할 수 있다

```
n = int(input('수를 입력하세요 :'))  
is_prime = True  
for num in range(2, n):           # 2부터 (n-1) 사이의 수 num에 대하여  
    if n % num == 0:              # 이 수 중에서 n의 약수가 있으면  
        is_prime = False         # 소수가 아님  
print(n, 'is prime :', is_prime)
```

# 2부터 100까지의 소수 구하기

코드 3-43 : 2부터 100까지의 소수 구하기

get\_primes.py

# 소수를 담을 리스트 초기화

primes = []

for n in range(2, 101):

# 일단 n을 소수라고 두자

is\_prime = True

for num in range(2, n): # 2~(n-1) 사이의 수 num에 대하여

if n % num == 0: # 이 수 중에서 n의 약수가 있으면

is\_prime = False # 소수가 아님

if is\_prime: # 소수일 경우 primes라는 리스트에 추가한다

primes.append(n) # append() 메소드는 리스트에 n을 추가함

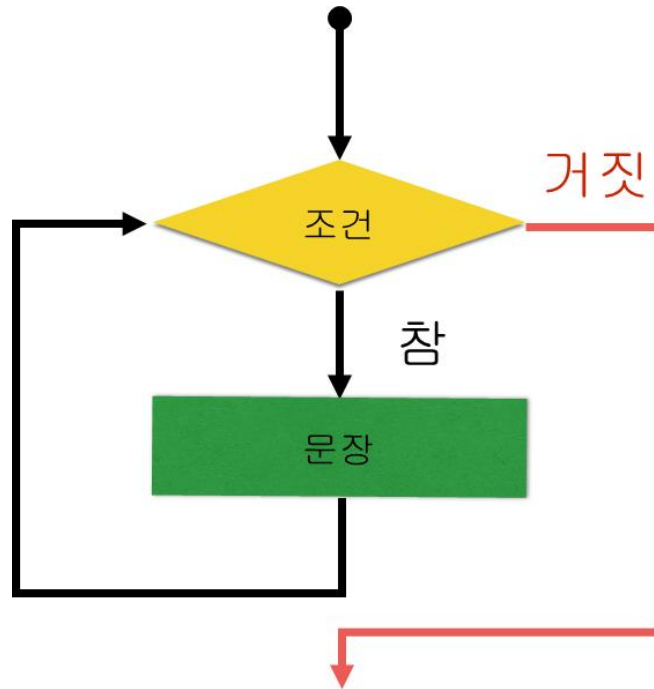
print(primes)

실행결과

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,  
37, 41, 43, 47, 53, 59, 61, 67, 71, 73,  
79, 83, 89, 97]

## 3.7 while 반복문

- 3.7.1 while 반복문의 구조
  - 조건이 참인 경우에 계속 실행하는 반복문



# while문의 문법

- if문과 매우 유사
- 조건식이 참이라면 계속 반복하여 해당 코드를 실행

형 식	예 시
초기값 지정 while 조건식 : 실행할 코드 블록	<pre>i = 0          # 초기 값 지정 while i&lt;5 :    # 조건식     print('Welcome to everyone!!')     i += 1</pre>

코드 3-44 : for 문을 이용한 'Welcome to everyone!!'의 반복 출력기능

print\_welcome\_with\_for.py

```
for i in range(5):  
    print('Welcome to everyone!!')
```

코드 3-45 : while 문을 이용한 'Welcome to everyone!!'의 반복 출력기능

print\_welcome\_with\_while.py

```
i = 0 # 초기 값  
while i < 5: # 루프의 조건식이 참이면 내부 블록이 실행됨  
    print('Welcome to everyone!!')  
    i += 1 # 조건 값의 변경
```

실행결과

```
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!  
Welcome to everyone!!
```

- i를 0부터 1씩 증가시키며 5보다 작을 때까지 while 내부의 코드를 반복함
- 다음과 같이 새로 작성 가능

코드 3-46 : 지정된 수까지의 누적 합을 구하는 기능

while\_sum\_input.py

```
n = int(input('합계를 구할 수를 입력하세요 : '))
s = 0
i = 1
while i <= n:
    s = s + i
    i += 1
print('1부터 {}까지의 합은 {}'.format(n, s))
```

실행결과

합계를 구할 수를 입력하세요 : 100

1부터 100까지의 합은 5050

- 반복실행의 횟수가 명확한 경우는 while 문의 코드가 길어지기 때문에 for문을 사용하는 것이 더 나은 방법



# while문과 for문 비교

- while문은 수행횟수를 정확히 모르지만 수행의 조건이 명확한 경우에 더 적합
- 반복 횟수가 명확한 경우 for문이 적합

1에서 n까지의 합을 구하는 코드 비교	
while 문	for 문
<pre>s = 0 i = 1 while i &lt;= n:     s = s + i     i += 1</pre>	<pre>s = 0  for i in range(1, n+1) :     s = s + i</pre>

## 3.7.2 while 반복문과 입력조건

- 사용자로부터 '가위', '바위', '보'를 입력으로 받아서 이 값을 출력하는 게임 프로그램 만들기

코드 3-47 : while 반복문을 이용한 가위, 바위, 보 선택하기

rsp\_input.py

```
selected = None
while selected not in ['가위', '바위', '보']:
    selected = input('가위, 바위, 보 중에서 선택하세요> ')
print('선택한 값은:', selected)
```

#### 실행결과

```
가위, 바위, 보 중에서 선택하세요> 묵
가위, 바위, 보 중에서 선택하세요> 찌
가위, 바위, 보 중에서 선택하세요> 빠
가위, 바위, 보 중에서 선택하세요> 가위
선택한 값은: 가위
```

- 원하는 값 입력시  
실행되도록 하는  
반복문에 적합함

코드 3-48 : 양수 n을 입력받아 1부터 n까지의 합을 구하는 코드

for\_sum\_input2.py

```
n = int(input('합계를 구할 양의 정수를 입력하세요 : '))
```

```
s = 0
```

```
for i in range(1, n+1) :
```

```
    s = s + i
```

```
print('1부터 {}까지의 합은 {}'.format(n, s))
```

실행결과

합계를 구할 양의 정수를 입력하세요 : -10

1부터 -10까지의 합은 0

- 입력 값의 범위를 양의 자연수로 한정하려고 할 경우에도 while 문을 사용하는 것이 적합

코드 3-49 : 1부터 n까지의 합을 구하는 코드로 while 문 내에서 입력문 사용

for\_sum\_input2\_1.py

```
n = -1
while n <= 0: # 양수가 입력될 때 까지 input() 문을 반복 수행함
    n = int(input('합계를 구할 양의 정수를 입력하세요 : '))
s = 0
for i in range(1, n+1):
    s = s + i
print('1부터 {}까지의 합은 {}'.format(n, s))
```

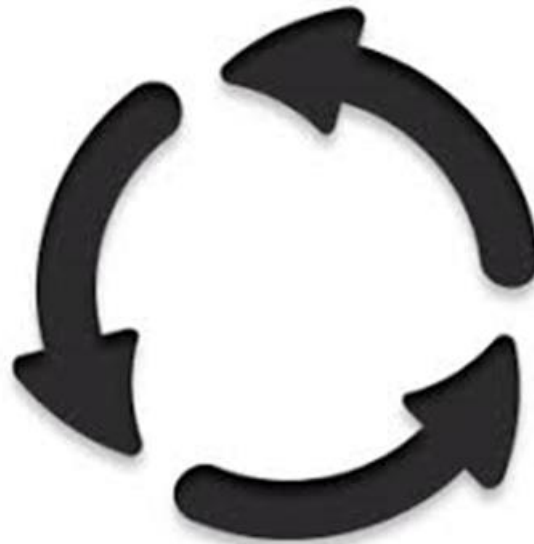
#### 실행결과

```
합계를 구할 양의 정수를 입력하세요 : -10
합계를 구할 양의 정수를 입력하세요 : 0
합계를 구할 양의 정수를 입력하세요 : 10
1부터 10까지의 합은 55
```

- $n \leq 0$ 이라는 조건을 만족할 경우 재입력을 받는 부분이 핵심

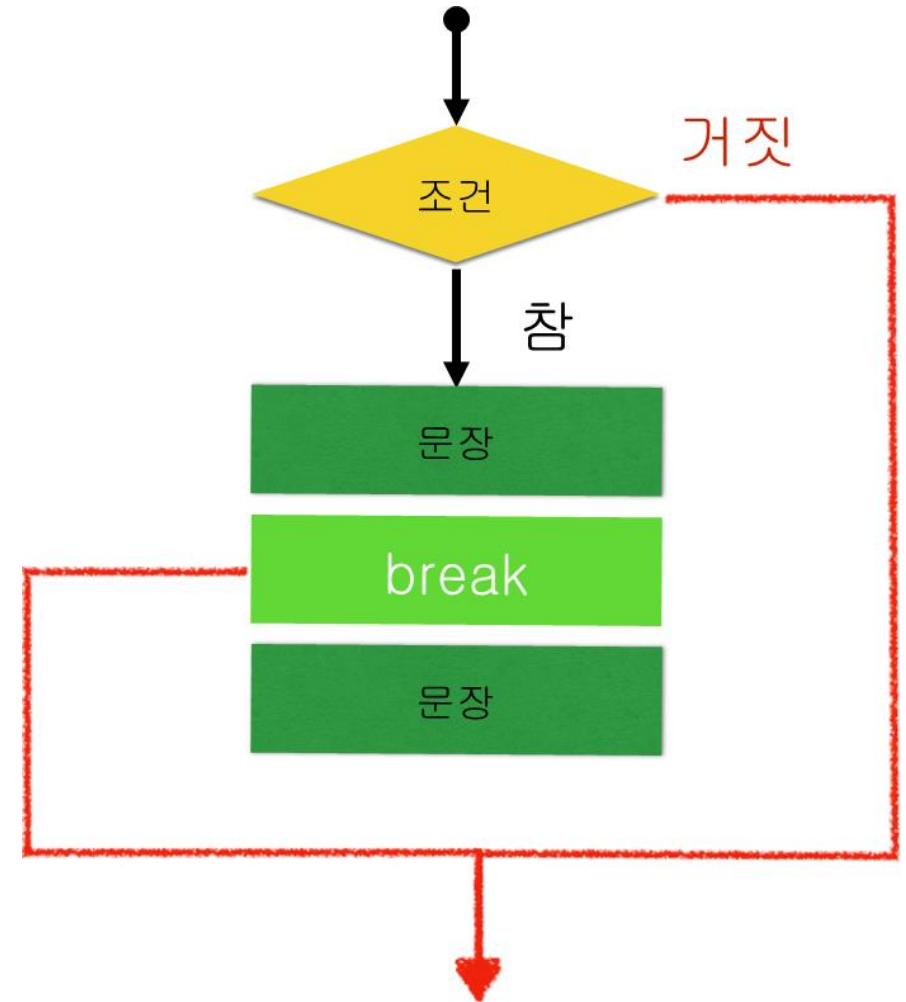
## 3.8 break와 continue

- 반복문을 제어하는 키워드
  - 반복 실행을 종료 -> break
  - 반복문 루프 내의 나머지 실행부를 건너뛰고 계속해서 반복 루프를 실행 -> continue
  - continue는 반복 실행을 종료하지 않음



# break를 표현한 흐름도

- while이나 반복문은 조건이 참이면 블록내의 문장을 수행
- 도중에서 break를 만나면 그 즉시 반복 실행을 종료하고 루프를 빠져나옴



코드 3-50 : break를 사용하여 모음이 나타나면 즉시 반복문을 종료하는 기능

```
skip_vowel_break.py
```

```
st = 'Programming'
```

```
# 자음이 나타나는 동안만 출력하는 기능
```

```
for ch in st:
```

```
    if ch in ['a','e','i','o','u']:
```

```
        break    # 모음일 경우 반복문을 종료한다.
```

```
    print(ch)
```

```
print('The end')
```

실행결과

P

r

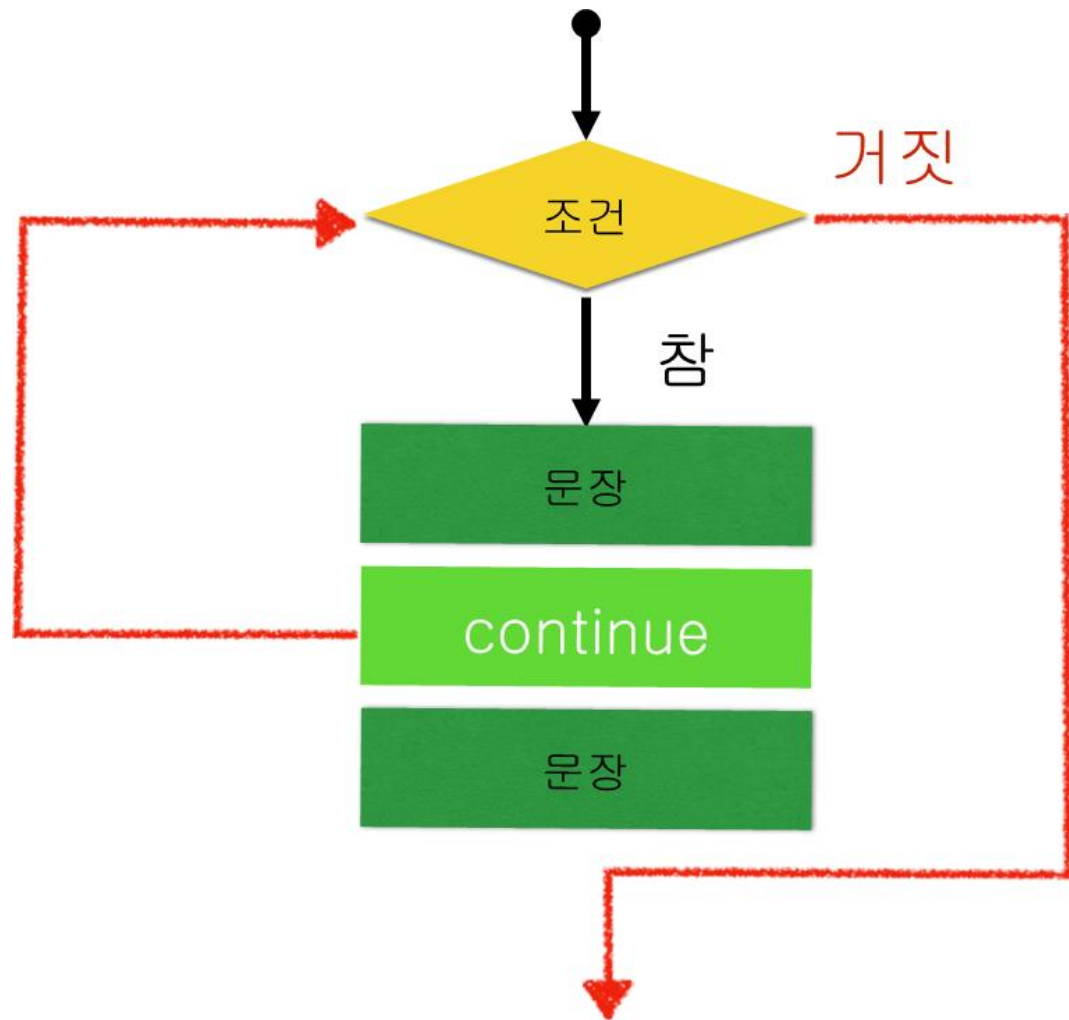
The end

- 모음에 해당하면 break
- 그렇지 않으면 print를 이용하여 출력
- break를 작동시키면 반복문의 나머지 부분을 실행하지 않고 루프를 중지시킴



# continue를 표현한 흐름도

- 루프를 빠져나오지 않고 continue 아래의 문장만을 건너뛰는 역할
- 반복문이 종료되는 것은 조건이 거짓일 때에만 해당



코드 3-51 : continue를 사용하여 모음일 경우 출력을 건너뛰는 기능

skip\_vowel\_continue.py

```
st = 'Programming'
```

```
# 자음이 나타날때만 출력하는 기능
```

```
for ch in st:
```

```
    if ch in ['a','e','i','o','u']:
```

```
        continue    # 모음일 경우 아래 출력을 건너뛴다
```

```
    print(ch)
```

```
print('The end')
```

실행결과

P

r

g

r

m

m

n

g

The end

- continue를 넣게 되면 아래에 있는 아래의 나머지 부분을, 즉 print를 실행하지 않고 반복문의 처음으로 돌아가는 기능을 함



### 주의 : break와 continue의 흐름제어의 위험성

break와 continue는 프로그램의 제어를 효율적으로 하는데 편리하게 사용할 수 있다. 하지만, break와 continue 문이 너무 많이 사용되는 경우 제어의 흐름에 일관성이 없어 프로그램을 이해하는 것이 어려워진다. 따라서 continue와 break는 필요한 경우에만 제한적으로 사용하는 것이 좋다.

# 할 일

- LAB 문제를 하나하나 풀어봅시다.
- 3장의 연습문제를 하나씩 풀어보며 3장의 내용을 복습해 봅시다.



Questions?