

프로그래밍 언어, 파이썬

- 파이썬 (Python)

- 네덜란드의 귀도 반 로섬 (Guido van Rossum)이 개발

- 1989년 크리스마스가 있던 주에 자신이 출근하던

- 연구실의 문이 닫혀 있어서 취미 삼아 만들었으며,

- 이후 개발을 거듭하여 1991년에 파이썬을 외부에 공개

- 파이썬이라는 이름은 자신이 좋아하는 코미디 쇼인

- ‘몬티 파이썬의 날아다니는 서커스 (Monty Python’s Flying Circus)’

- 에서 따 옴

- Python의 원래 뜻이 비단뱀이라 로고와 아이콘이 뱀모양



프로그래밍 언어, 파이썬

- 파이썬의 장점

- 비전공자도 쉽게 배울 수 있음
- 다양한 분야에서 활용할 수 있음
- 대부분의 운영체제에서 동일하게 사용됨



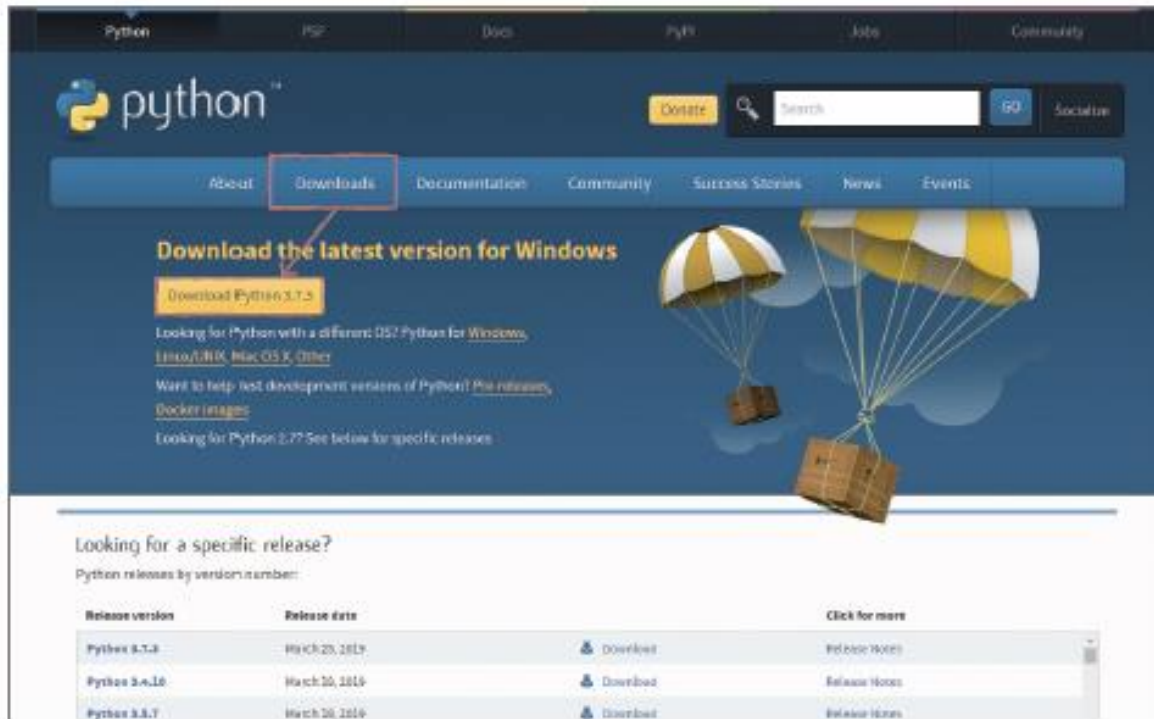
- 파이썬의 단점

- C언어에 비해 일반적으로 10~350배 느림
- 최근에는 컴퓨터 성능이 좋아져 연산이 많이 필요한 프로그램이 아니라면 차이 크게 느낄 수 없음

파이썬 설치하기

- 파이썬 설치 프로그램 다운로드

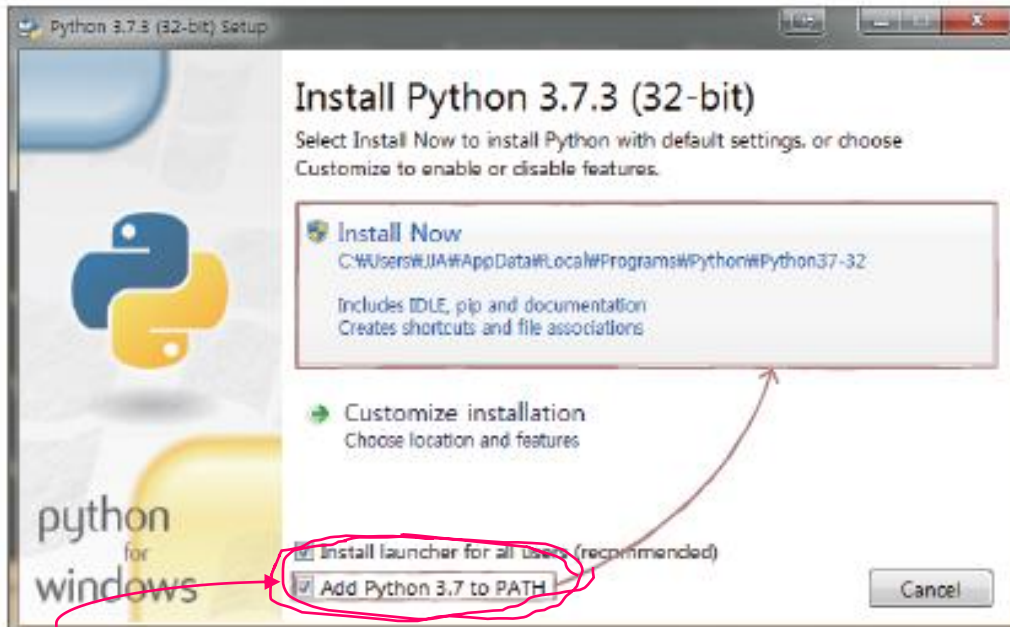
- 1) 파이썬 공식 홈페이지 (<http://www.python.org>) 접속 – [Downloads]
- 2) [Download Python 3.7.4] 클릭



파이썬 설치하기

- 파이썬 설치하기

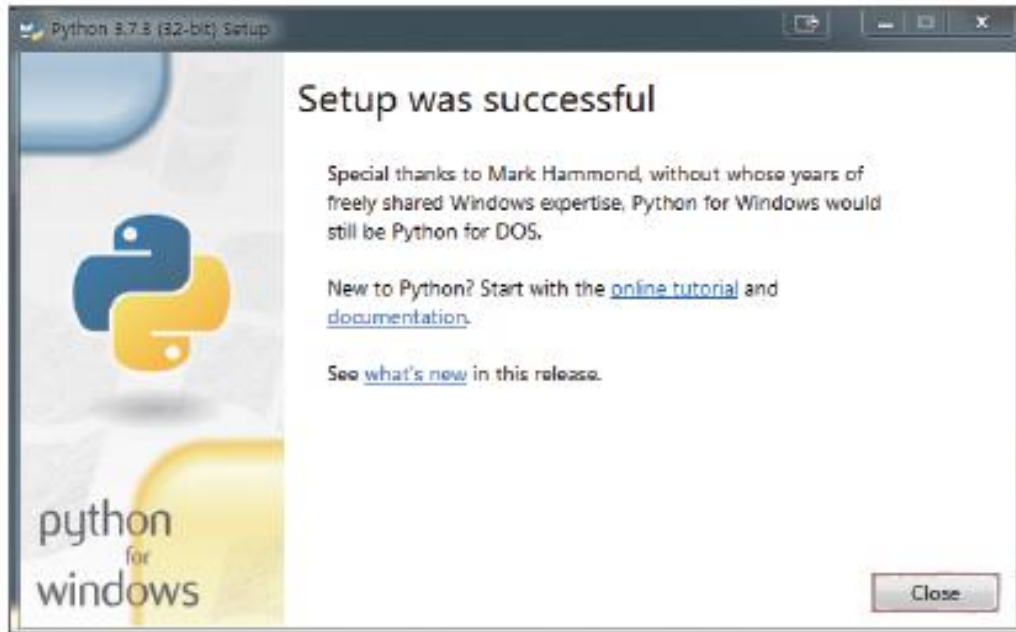
- 1) 설치 프로그램 실행하여 아래 화면에서 [Add Python 3.7 to PATH] 체크
- 2) [Install Now] 클릭



여기를 체크하지 않고 설치하면 파이썬이 실행되지 않으므로 꼭 확인합니다.

파이썬 설치하기

3) 설치 완료 화면이 나타나면 [Close] 클릭



4) 윈도우 [시작] 메뉴에서 [Python 3.7] 프로그램 확인



파이썬 실행하기 : 파이썬 인터랙티브 셸

- 인터프리터 (interpreter)
 - 파이썬으로 작성된 코드를 실행해주는 프로그램
- 파이썬 인터랙티브 셸
 - 파이썬 명령어를 한 줄씩 입력하며 실행결과 볼 수 있는 공간

```
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 22:20:52) [MSC v.1916 32 bit  
(Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

파이썬 실행하기 : 파이썬 인터랙티브 셸

- 프롬프트 (prompt)

- >>>

- 코드를 한 줄씩 입력

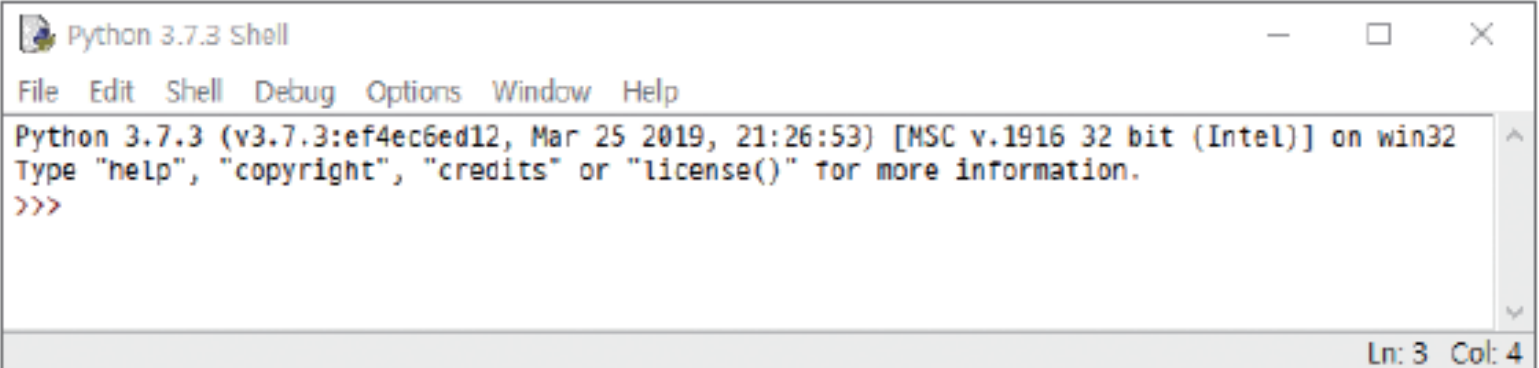
- 인터랙티브 셸 = 대화형 셸

: 컴퓨터와 상호 작용하는 공간이며, 한 마디씩 주고받는 것처럼 대화한다고 하여 대화형 셸로 불리기도 함

```
>>> 10 + 10 Enter → 10 + 10을 입력하니
20 → 10과 10을 더해 20을 출력합니다.
>>> "Hello" * 3 Enter → Hello라는 문자열을 3번 출력하라는 의미이며,
'HelloHelloHello' → 'HelloHelloHello'를 출력합니다.
>>>
```

텍스트 에디터 사용하기 : 파이썬 IDLE 에디터

- 텍스트 에디터 (text editor)
 - 긴 코드를 입력하거나 코드를 저장해야 하는 경우 사용
 - 글자를 적을 수 있는 모든 종류의 프로그램
 - IDLE 에디터에서 코드 작성하고 실행하기
 - 파이썬은 통합 개발 환경으로써 IDLE 제공
- 1) [시작 메뉴] – [Python 3.7] – [IDLE]

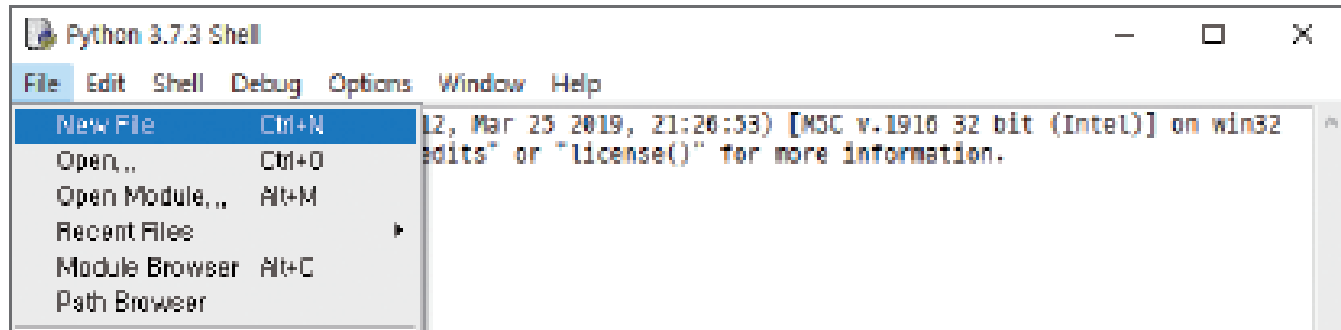


```
Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
```

Ln: 3 Col: 4

텍스트 에디터 사용하기 : 파이썬 IDLE 에디터

2) [File] – [New File] 메뉴 선택

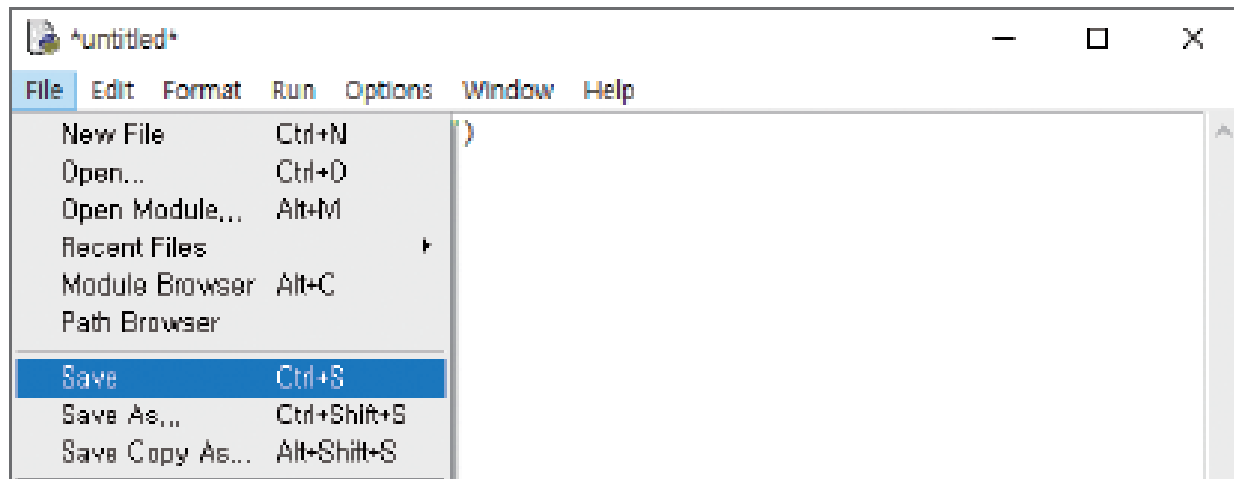


3) 아래와 같이 입력

```
print("IDLE에서 파이썬 코드를")  
print("작성해서 출력해 보는")  
print("예제입니다")
```

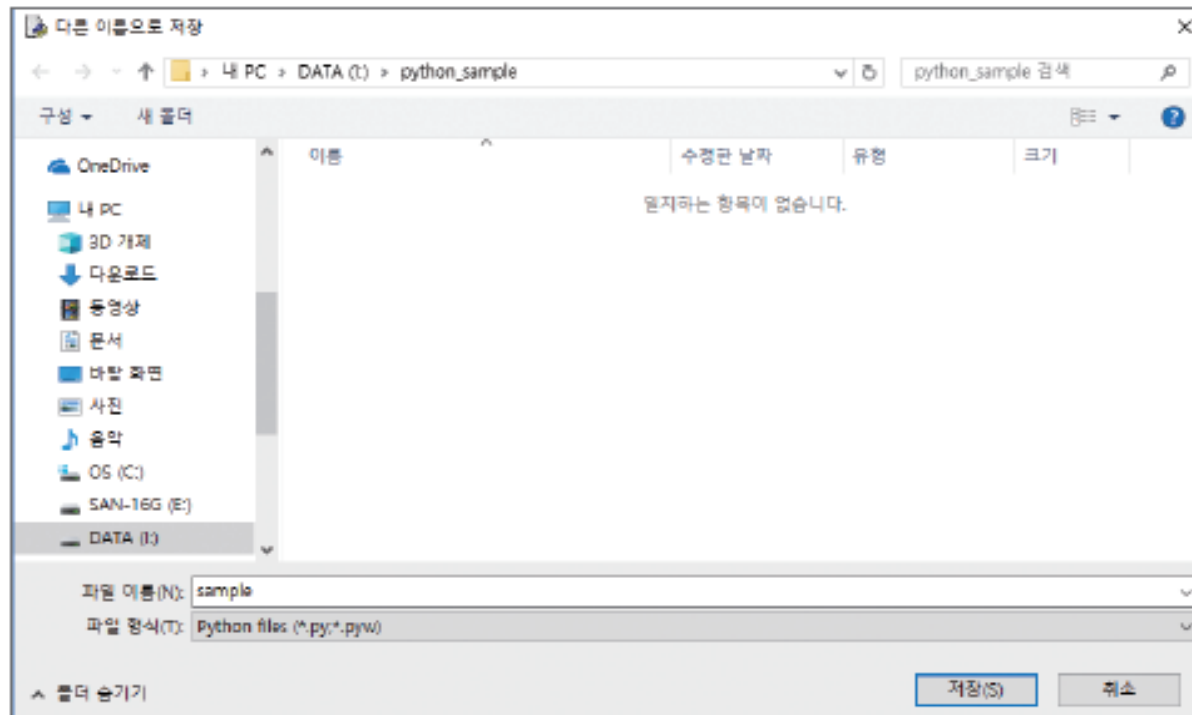
텍스트 에디터 사용하기 : 파이썬 IDLE 에디터

4) [File] – [Save] 메뉴 선택



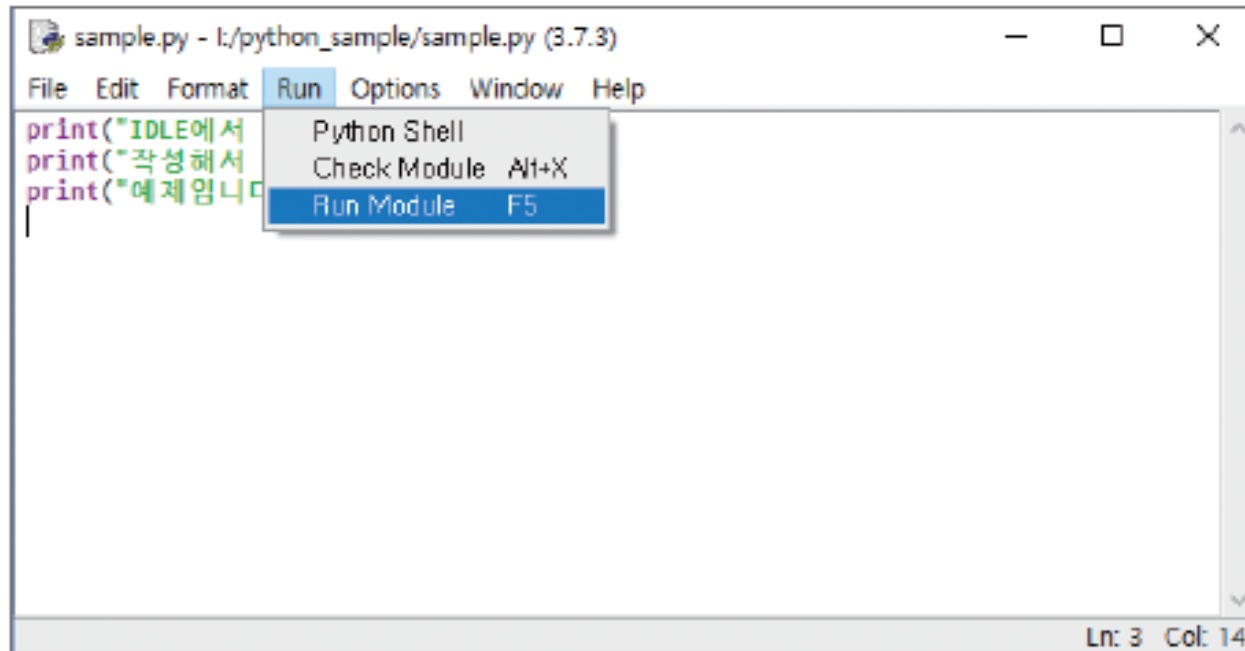
텍스트 에디터 사용하기 : 파이썬 IDLE 에디터

5) [다른 이름으로 저장] 대화상자에서 파일 이름을 'sample'로 저장



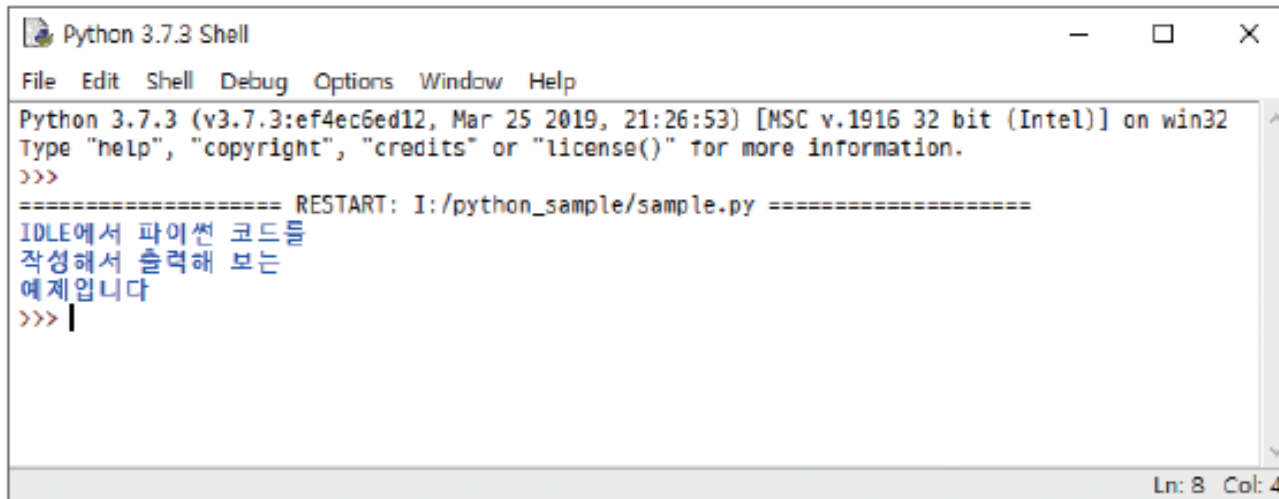
텍스트 에디터 사용하기 : 파이썬 IDLE 에디터

6) [Run] – [Run Module] 메뉴 선택(혹은 [F5] 단축키)



텍스트 에디터 사용하기 : 파이썬 IDLE 에디터

7) 파이썬 코드가 실행됨



The screenshot shows a window titled "Python 3.7.3 Shell" with a menu bar (File, Edit, Shell, Debug, Options, Window, Help). The main text area displays the following content:

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: I:/python_sample/sample.py =====
IDLE에서 파이썬 코드들
작성해서 출력해 보는
예제입니다
>>> |
```

The status bar at the bottom right indicates "Ln: 8 Col: 4".

주석

- 주석 (comment)
 - 프로그램 진행에 영향 주지 않는 코드
 - 프로그램 설명 위해 사용
 - # 기호를 주석으로 처리하고자 하는 부분 앞에 붙임

```
>>> # 간단히 출력하는 예입니다.
>>> print("Hello! Python Programming...") # 문자열을 출력합니다.
Hello! Python Programming...
```

→ # 기호 뒷부분이 주석 처리됩니다.

출력 : print()

- print() 함수
 - 출력 기능
 - 출력하고 싶은 것들을 괄호 안에 나열

```
print(출력1, 출력2, ...)
```

- 하나만 출력하기

```
>>> print("Hello! Python Programming...")
Hello! Python Programming...
>>> print(52)
52
>>> print(273)
273
```

출력 : print()

- 여러 개 출력하기

```
>>> print(52, 273, "Hello")
52 273 Hello
>>> print("안녕하세요", "저의", "이름은", "윤인성입니다!")
안녕하세요 저의 이름은 윤인성입니다!
```

- 줄바꿈하기

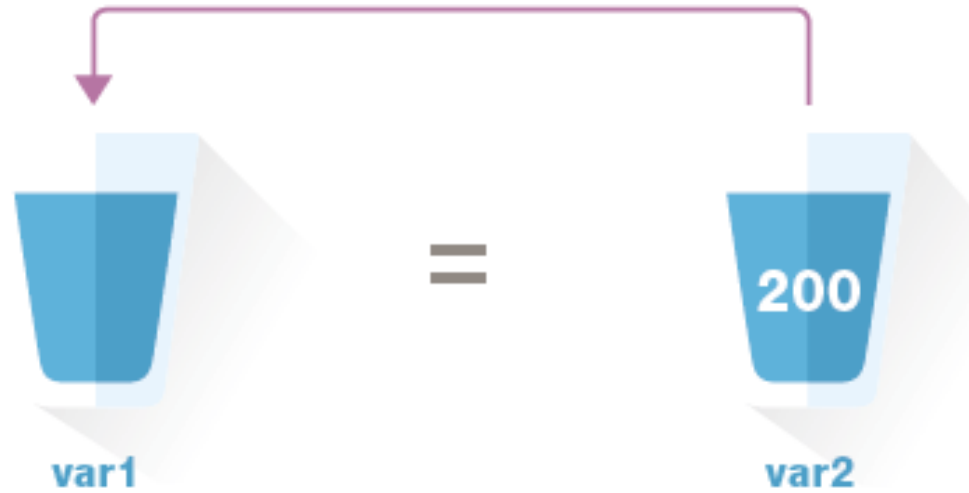
```
>>> print()
      → 빈 줄을 출력합니다.
>>>
```


변수



변수

var2 변수의 값(200)만 뽑아서
var1 변수에 대입



변수



변수

```
boolVar = False  
intVar = 100  
floatVar = 123.45  
strVar = "안녕?"
```

False



boolVar
불형 변수

100



intVar
정수형 변수

123.45



floatVar
실수형 변수

“안녕?”

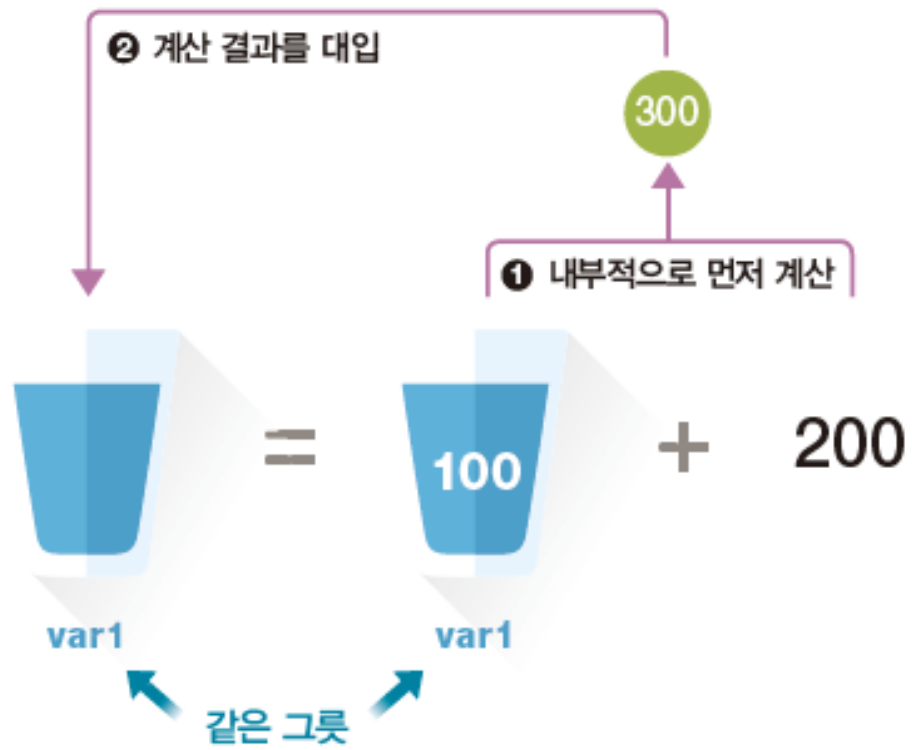


strVar
문자열 변수

변수



변수



변수



키워드(예약어)

| | | | | | |
|-------|--------|----------|-------|--------|----------|
| False | None | True | and | as | assert |
| break | class | continue | def | del | elif |
| else | except | finally | for | from | global |
| if | import | in | is | lambda | nonlocal |
| not | or | pass | raise | return | try |
| while | with | yield | | | |

식별자

- 스네이크 케이스와 캐멀 케이스

`itemlist`

`loginstatus`

`characterhp`

`rotateangle`

- 공백이 없어 이해하기 어려움

- 스네이크 케이스 (snake case) : 언더바(_)를 기호 중간에 붙이기

- 캐멀 케이스 (camel case) : 단어들의 첫 글자를 대문자로 만들기

| 식별자에 공백이 없는 경우 | 단어 사이에 _ 기호를 붙인 경우 (스네이크 케이스) | 단어 첫 글자를 대문자로 만든 경우 (캐멀 케이스) |
|---|---|---|
| <code>itemlist</code> <code>loginstatus</code> <code>characterhp</code> <code>rotateangle</code> | <code>item_list</code> <code>login_status</code> <code>character_hp</code> <code>rotate_angle</code> | <code>ItemList</code> <code>LoginStatus</code> <code>CharacterHp</code> <code>RotateAngle</code> |

- 파이썬에서는 스네이크 및 캐멀 케이스 둘 모두 사용

식별자

- 식별자 구분하기

- 캐멀 케이스에서는 첫 번째 글자를 소문자로 적지 않음

캐멀 케이스 유형 1 : `PrintHello` → 파이썬에서 사용합니다.

캐멀 케이스 유형 2 : `printHello` → 파이썬에서 사용하지 않습니다.

`print`

`input`

`list`

`str`

`map`

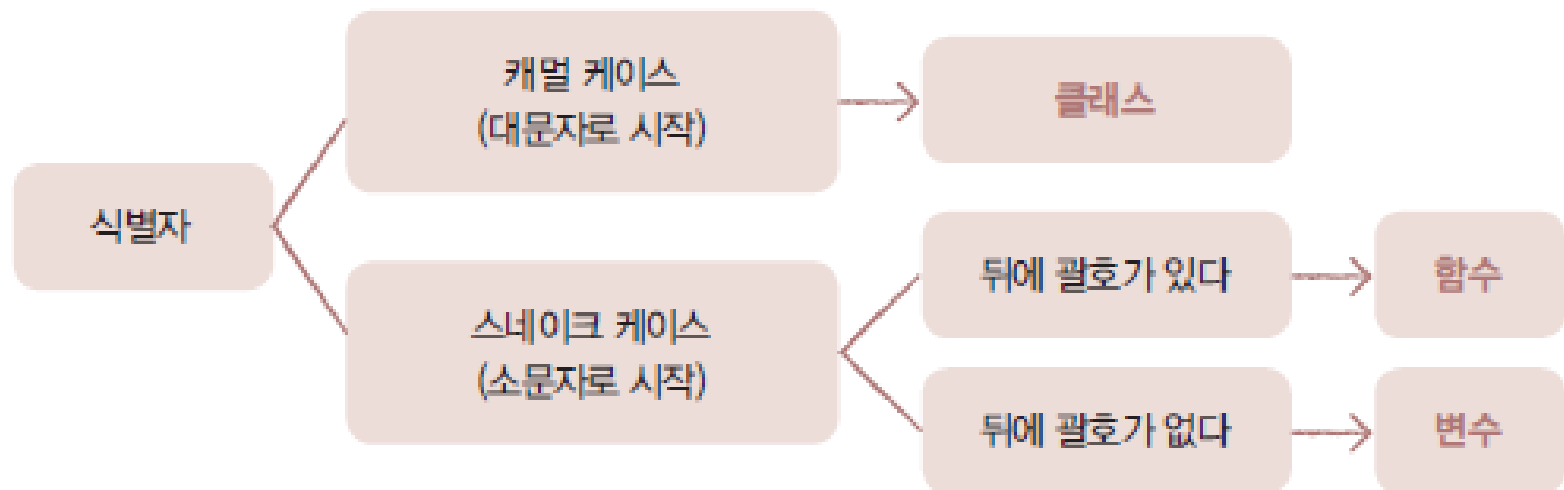
`filter`

`Animal`

`Customer`

식별자

- 캐멀 케이스로 작성되었으면 클래스
- 스네이크 케이스로 작성되어 있으면 함수 또는 변수
- 뒤에 괄호 붙으면 함수
- 뒤에 괄호 없으면 변수



문자열

"Hello" 'String' '안녕하세요' "Hello Python Programming"

하나만 출력합니다.

print("# 하나만 출력합니다.")

print("Hello Python Programming...!")

print()

여러 개를 출력합니다.

print("# 여러 개를 출력합니다.")

print(10, 20, 30, 40, 50)

print("안녕하세요", "저의", "이름은", "윤인성입니다!")

...

문자열 만들기

- 큰따옴표로 문자열 만들기

```
>>> print("안녕하세요")  
안녕하세요
```

- 작은따옴표로 문자열 만들기

```
>>> print('안녕하세요')  
안녕하세요
```


문자열

"안녕하세요"라고 말했습니다

출력할 큰따옴표

```
>>> print("안녕하세요"라고 말했습니다)
```

문자열을 만들기 위해 사용한 큰따옴표

 오류

SyntaxError: invalid syntax

문자열

```
>>> print('안녕하세요'라고 말했습니다)
안녕하세요'라고 말했습니다
```

```
>>> print("'배가 고픈다'라고 생각했습니다")
'배가 고픈다'라고 생각했습니다
```

이스케이프 문자 (escape character)

```
>>> print("\"안녕하세요\"라고 말했습니다")
```

```
"안녕하세요"라고 말했습니다
```

```
>>> print('\'배가 고픈다\'라고 생각했습니다')
```

```
'배가 고픈다'라고 생각했습니다
```

```
>>> print("안녕하세요\n안녕하세요")
```

```
안녕하세요
```

```
안녕하세요
```

```
>>> print("안녕하세요\t안녕하세요")
```

```
안녕하세요
```

```
안녕하세요
```


이스케이프 문자 (escape character)

```
>>> print("동해물과 백두산이 마르고 닳도록\n하느님이 보우하사 우리나라 만세\n무궁화 삼천리 화려강  
산 대한사람\n대한으로 길이 보전하세")
```

동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려강산 대한사람
대한으로 길이 보전하세

이스케이프 문자 (escape character)

```
>>> print("""동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세""")  
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

이스케이프 문자 (escape character)

```
>>> print("""
```

```
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세  
""")
```

```
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

→ 위 아래로 의도하지 않은 줄바꿈이 들어갑니다.

이스케이프 문자 (escape character)

```
>>> print("""\
```

```
동해물과 백두산이 마르고 닳도록
```

```
하느님이 보우하사 우리나라 만세
```

```
무궁화 삼천리 화려강산 대한사람
```

```
대한으로 길이 보전하세\
```

```
""")
```

```
동해물과 백두산이 마르고 닳도록
```

```
하느님이 보우하사 우리나라 만세
```

```
무궁화 삼천리 화려강산 대한사람
```

```
대한으로 길이 보전하세
```

→ 줄을 바꿔 출력하지 않겠다고 선언합니다.

문자열 인덱싱(indexing)

| | | | | |
|-----|-----|-----|-----|-----|
| 안 | 녕 | 하 | 세 | 요 |
| [0] | [1] | [2] | [3] | [4] |

| | | | | |
|------|------|------|------|------|
| 안 | 녕 | 하 | 세 | 요 |
| [-5] | [-4] | [-3] | [-2] | [-1] |

문자열 슬라이싱 (slicing)

| | | | | |
|-----|-----|-----|-----|-----|
| 안 | 녕 | 하 | 세 | 요 |
| [0] | [1] | [2] | [3] | [4] |

산술 연산자

| 연산자 | 의미 | 사용 예 | 설명 |
|-----|--------|--------------|-------------------------------|
| = | 대입 연산자 | $a = 3$ | 정수 3을 a에 대입 |
| + | 더하기 | $a = 5 + 3$ | 5와 3을 더한 값을 a에 대입 |
| - | 빼기 | $a = 5 - 3$ | 5에서 3을 뺀 값을 a에 대입 |
| * | 곱하기 | $a = 5 * 3$ | 5와 3을 곱한 값을 a에 대입 |
| / | 나누기 | $a = 5 / 3$ | 5를 3으로 나눈 값을 a에 대입 |
| // | 나누기(몫) | $a = 5 // 3$ | 5를 3으로 나눈 후 소수점을 버리고 값을 a에 대입 |
| % | 나머지값 | $a = 5 \% 3$ | 5를 3으로 나눈 후 나머지값을 a에 대입 |
| ** | 제곱 | $a = 5 ** 3$ | 5의 3제곱을 a에 대입 |

관계 연산자

| 연산자 | 의미 | 설명 |
|-----|---------|---------------|
| == | 같다. | 두 값이 동일하면 참 |
| != | 같지 않다. | 두 값이 다르면 참 |
| > | 크다. | 왼쪽이 크면 참 |
| < | 작다. | 왼쪽이 작으면 참 |
| >= | 크거나 같다. | 왼쪽이 크거나 같으면 참 |
| <= | 작거나 같다. | 왼쪽이 작거나 같으면 참 |

논리 연산자

| 연산자 | 의미 | 설명 |
|-----------|----------|----------------|
| and(논리곱) | ~이고, 그리고 | 둘 다 참이어야 참 |
| or(논리합) | ~이거나, 또는 | 둘 중 하나만 참이어도 참 |
| not(논리부정) | ~아니다, 부정 | 참이면 거짓, 거짓이면 참 |

논리 연산자의 활용

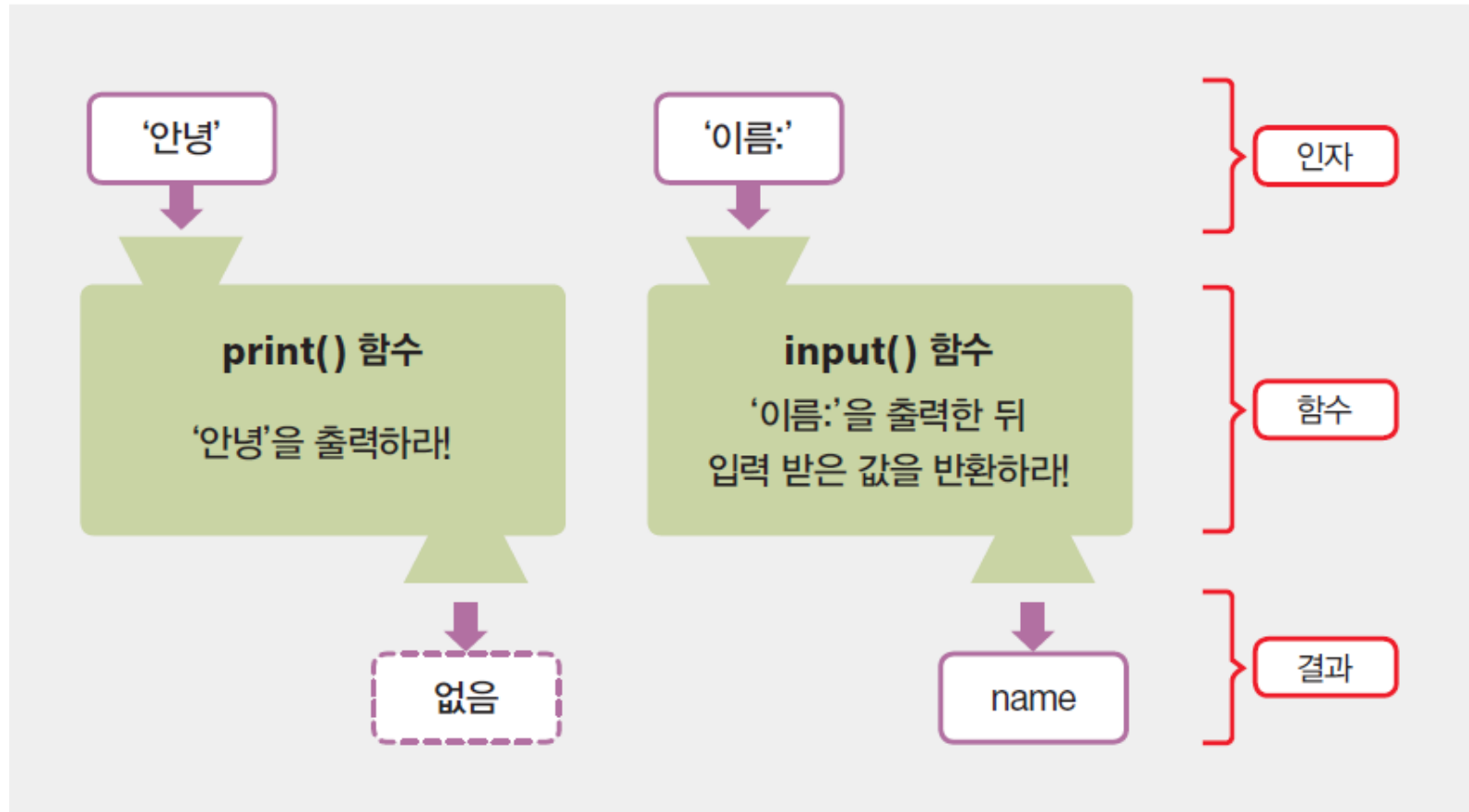
- and 연산자



- or 연산자



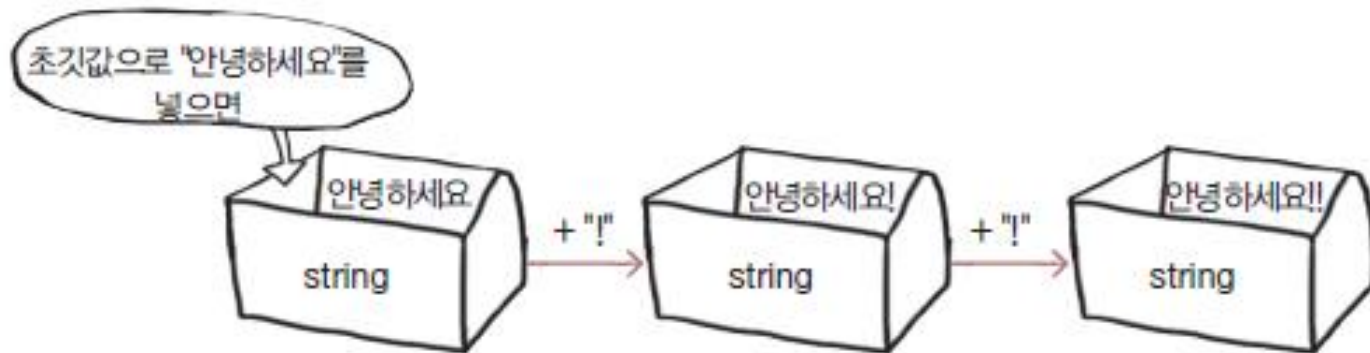
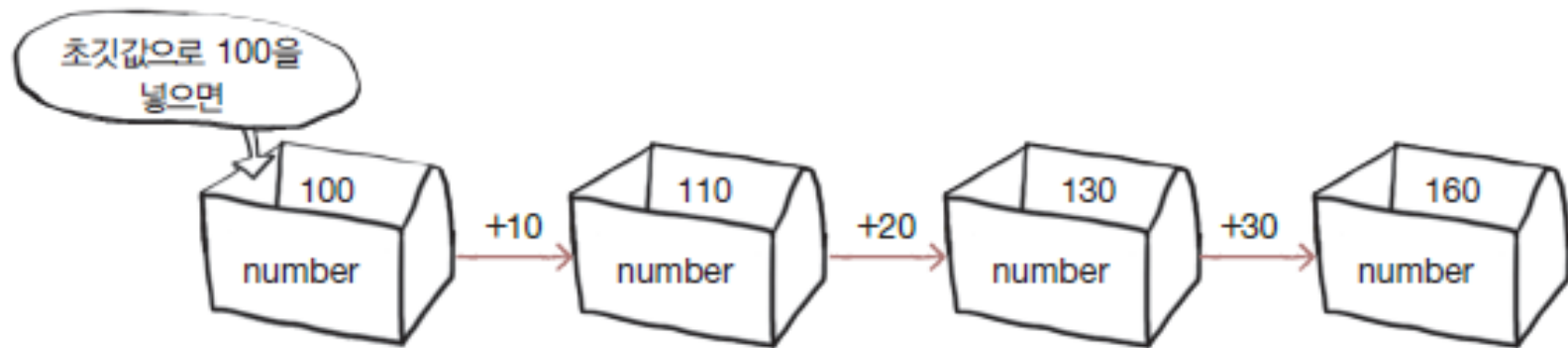
사용자 입력 : input()



복합 대입 연산자

| 연산자 | 사용 예 | 설명 |
|------------------|----------------------|------------------------------|
| <code>+=</code> | <code>a += 3</code> | <code>a = a + 3</code> 과 동일 |
| <code>-=</code> | <code>a -= 3</code> | <code>a = a - 3</code> 과 동일 |
| <code>*=</code> | <code>a *= 3</code> | <code>a = a * 3</code> 과 동일 |
| <code>/=</code> | <code>a /= 3</code> | <code>a = a / 3</code> 과 동일 |
| <code>//=</code> | <code>a //= 3</code> | <code>a = a // 3</code> 과 동일 |
| <code>%=</code> | <code>a %= 3</code> | <code>a = a % 3</code> 과 동일 |
| <code>**=</code> | <code>a **= 3</code> | <code>a = a ** 3</code> 과 동일 |

복합 대입 연산자



문자열의 길이 구하기

```
print(len("안녕하세요"))
```

↓ len("안녕하세요")은 5이므로,

```
print(5)
```

```
5
```

문자열과 in 연산자

```
>>> print("안녕" in "안녕하세요")  
True
```

```
>>> print("잘자" in "안녕하세요")  
False
```

문자열 자르기 : split()

```
>>> a = "10 20 30 40 50".split(" ")  
>>> print(a)  
['10', '20', '30', '40', '50']
```


문자열의 format() 함수

```
"{}".format(10)
```

```
"{} {}".format(10, 20)
```

```
"{} {} {} {} {}".format(101, 202, 303, 404, 505)
```

if 조건문이란

```
if x == 10:  
    print('10입니다.')
```

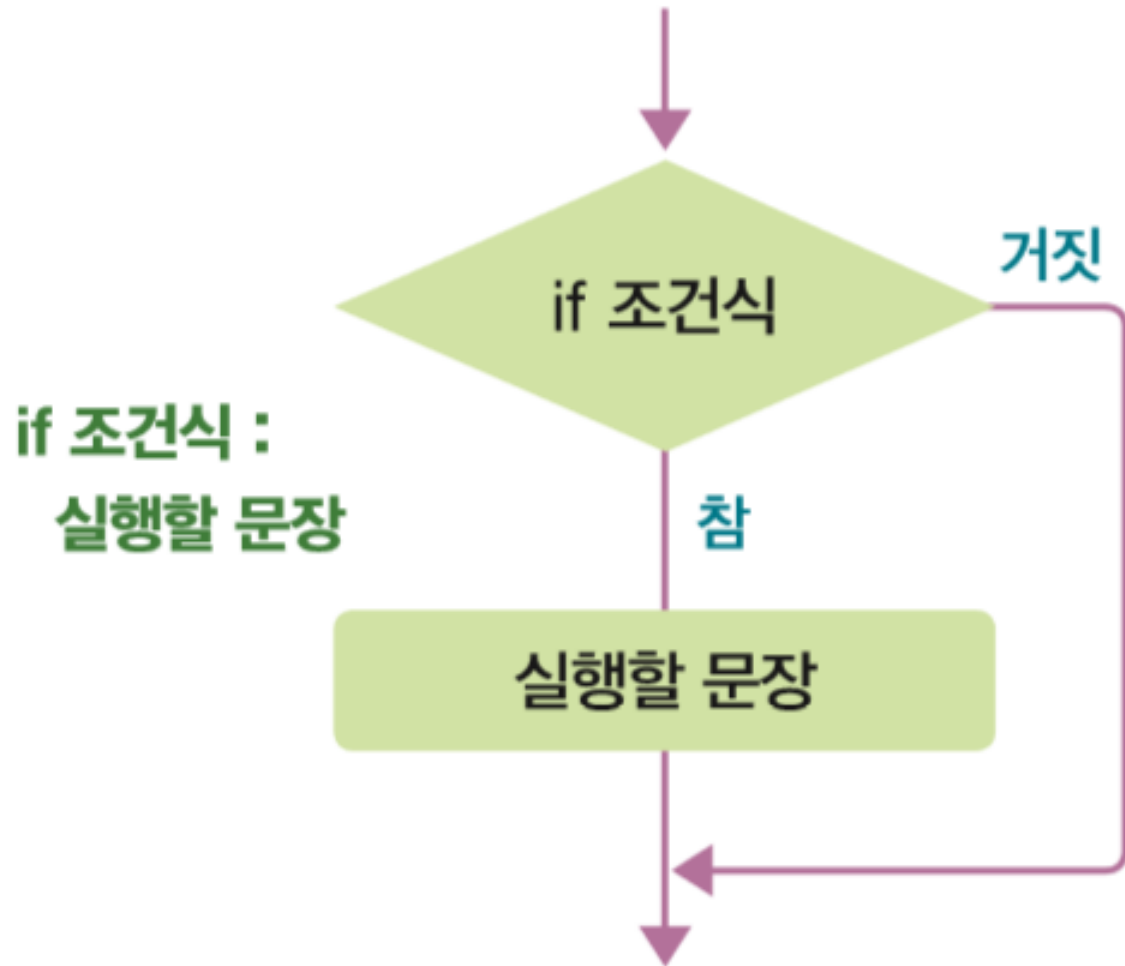
조건식

콜론

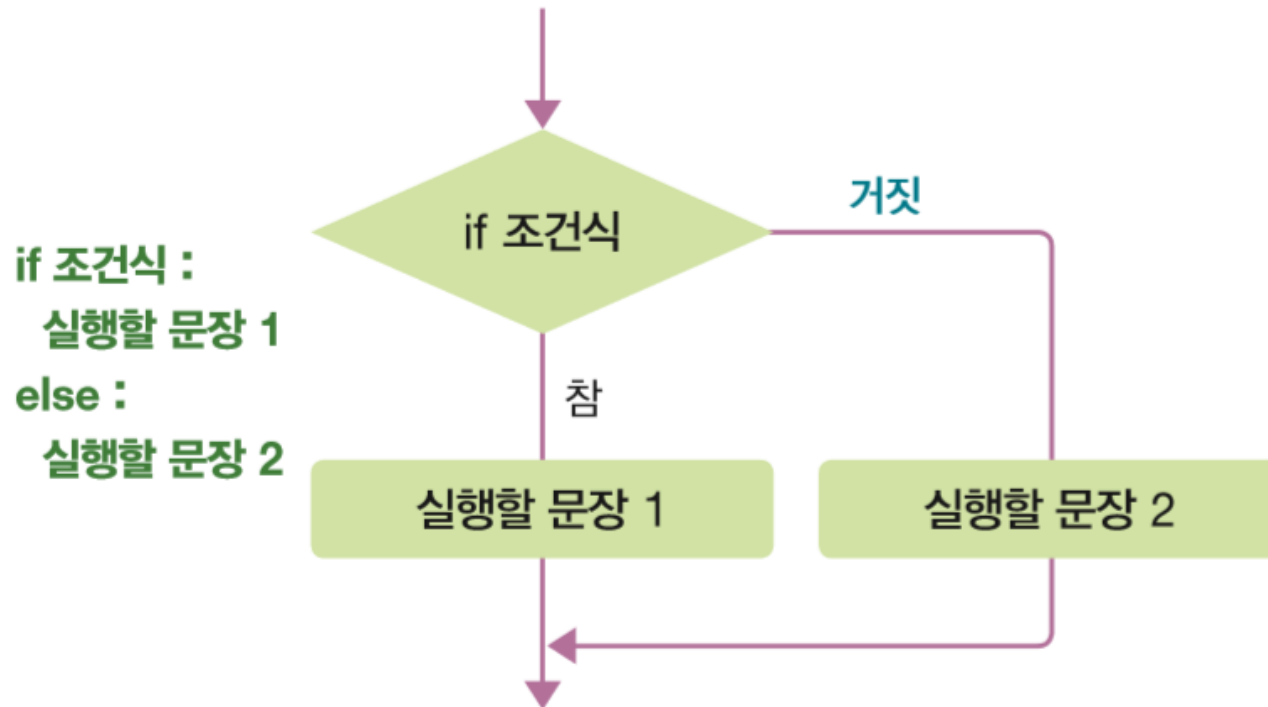
조건식이 만족할 때
실행할 코드

들여쓰기 4칸

if 조건문이란

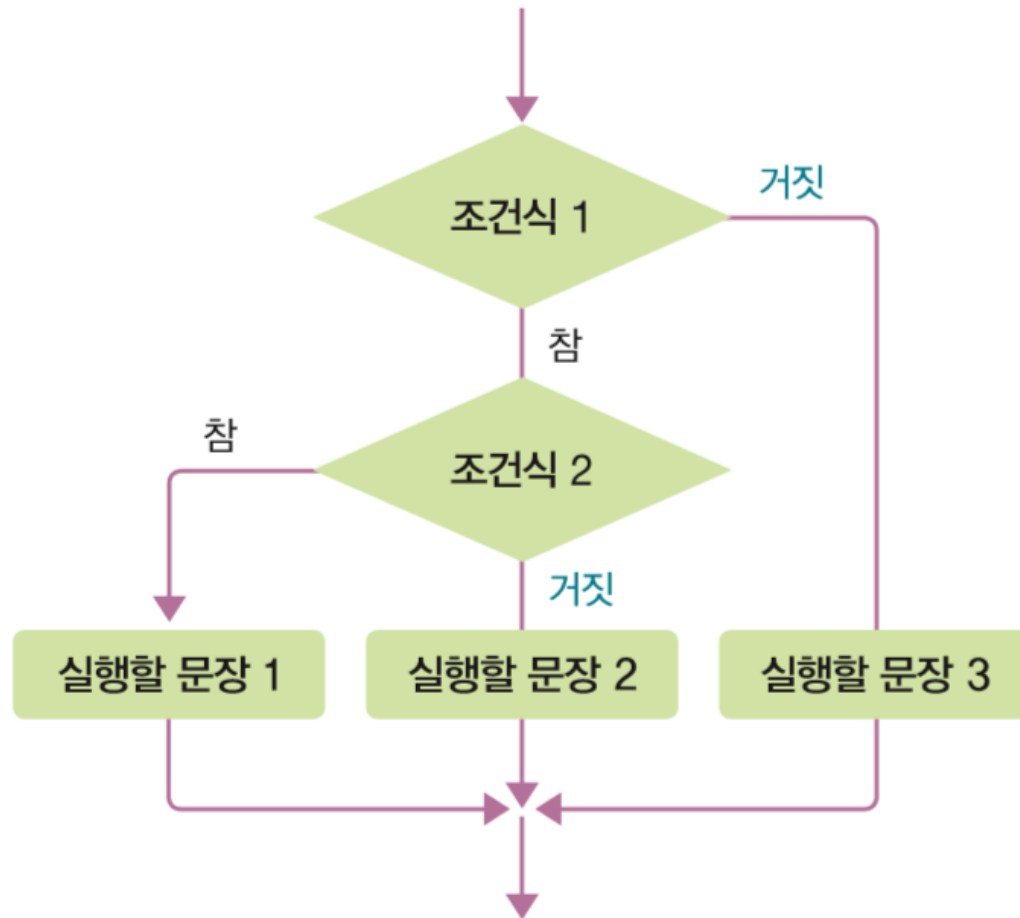


If ~ else 조건문

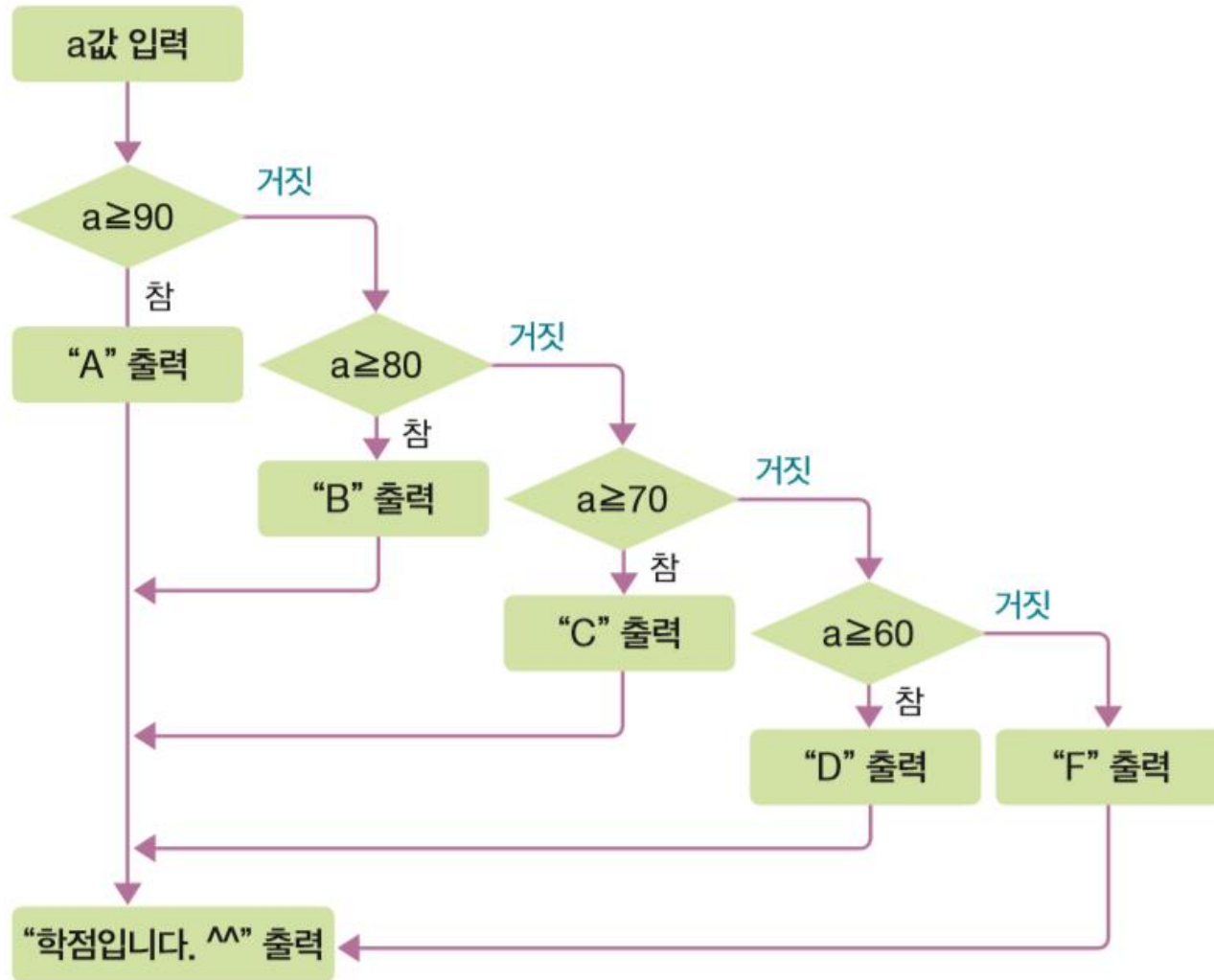


elif 구문

```
if 조건식 1 :  
    if 조건식 2 :  
        실행할 문장 1  
    else :  
        실행할 문장 2  
else :  
    실행할 문장 3
```



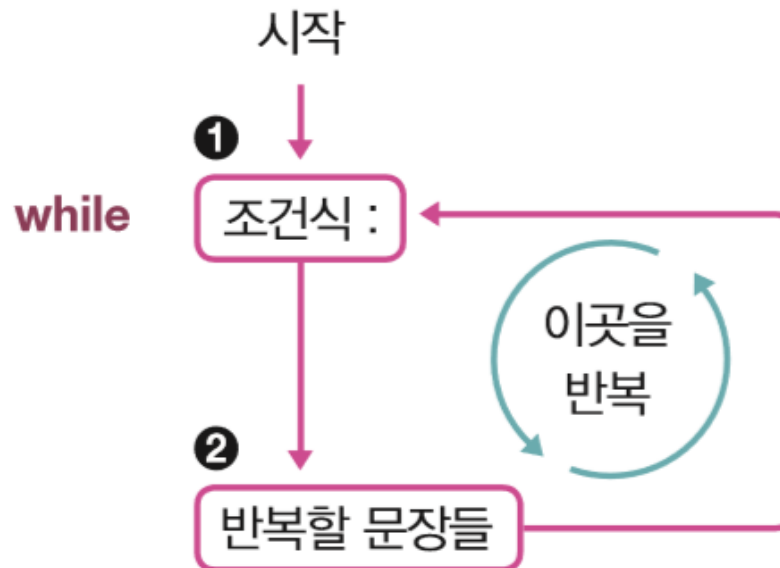
중첩 if ~ elif ~ else 구문



While 문

- For문과 While문의 비교

- for 문은 반복할 횟수를 range() 함수에서 결정 후 그 횟수만큼 반복, while 문은 반복 횟수를 결정하기보다는 조건식이 참일 때 반복하는 방식



While 문



기본 for문

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

range(3)은 range(0, 3, 1)과 같다

기본 for문

```
for 변수 in range( 종료 값 ) :  
    문장
```

0에서 (종료 값-1)까지의 숫자를 반환한다.

반복되는 문장으로
들여쓰기 하여야 한다.

Range() 함수

시작값이다.

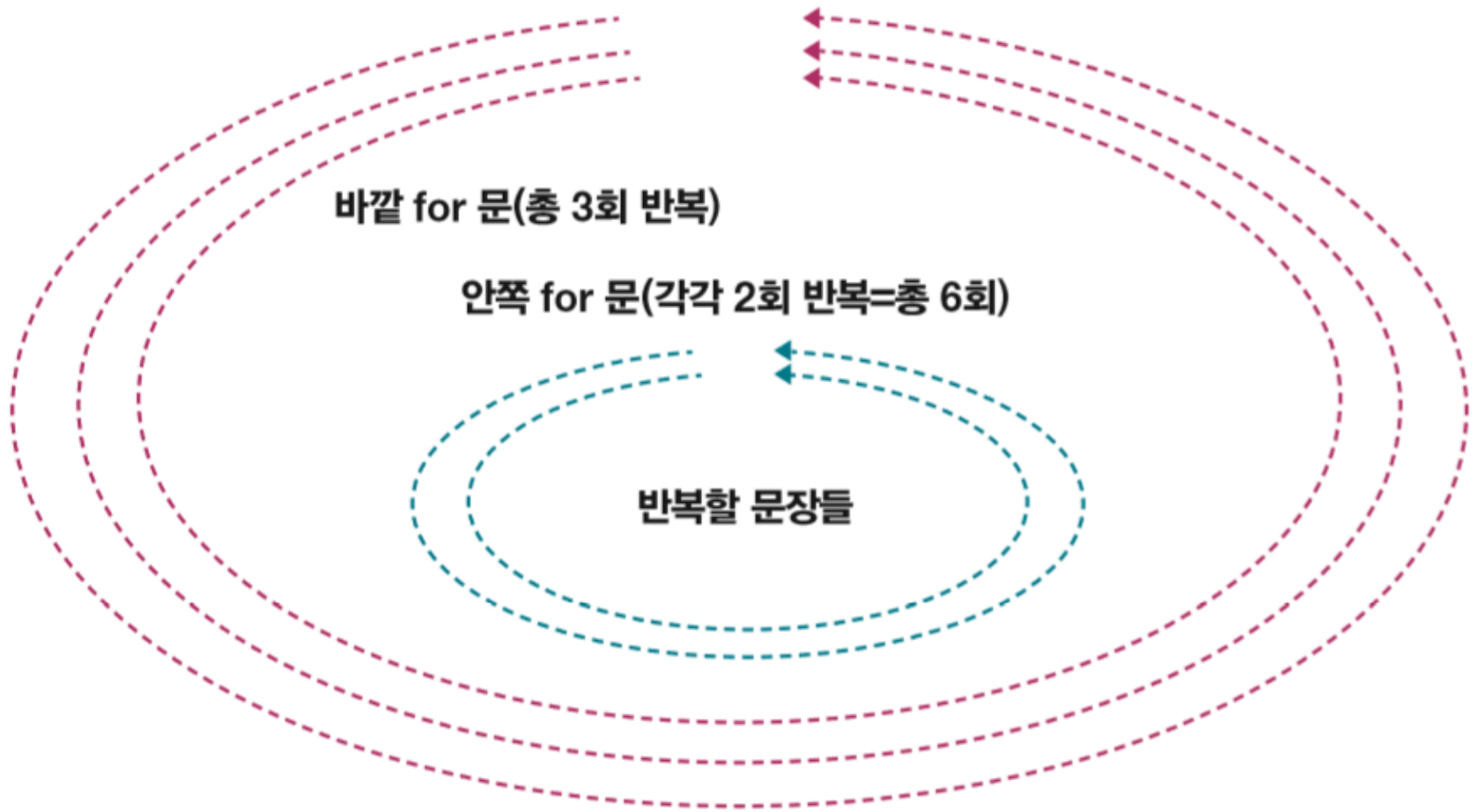
종료 값이지만 stop은
포함되지 않는다.

한 번에 증가되는 값이다.

`range(start=0, stop, step=1)`



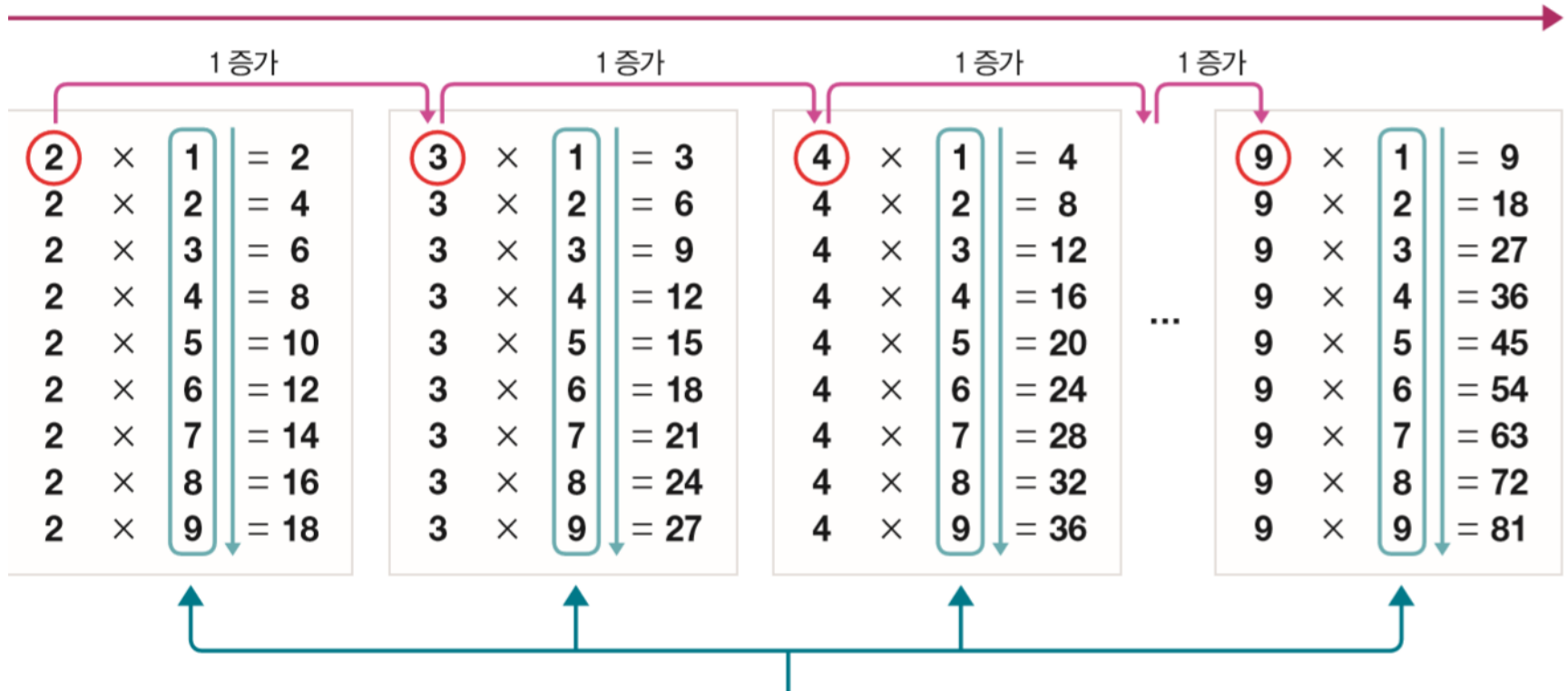
중첩 for문



$3 \times 2 = 6$ 번 반복

중첩 for문을 이용한 구구단 만들기

2에서 9까지 증가 후 종료(바깥 for 문 : i 변수)



1에서 9까지 계속 반복해서 증가(안쪽 for 문 : k 변수)

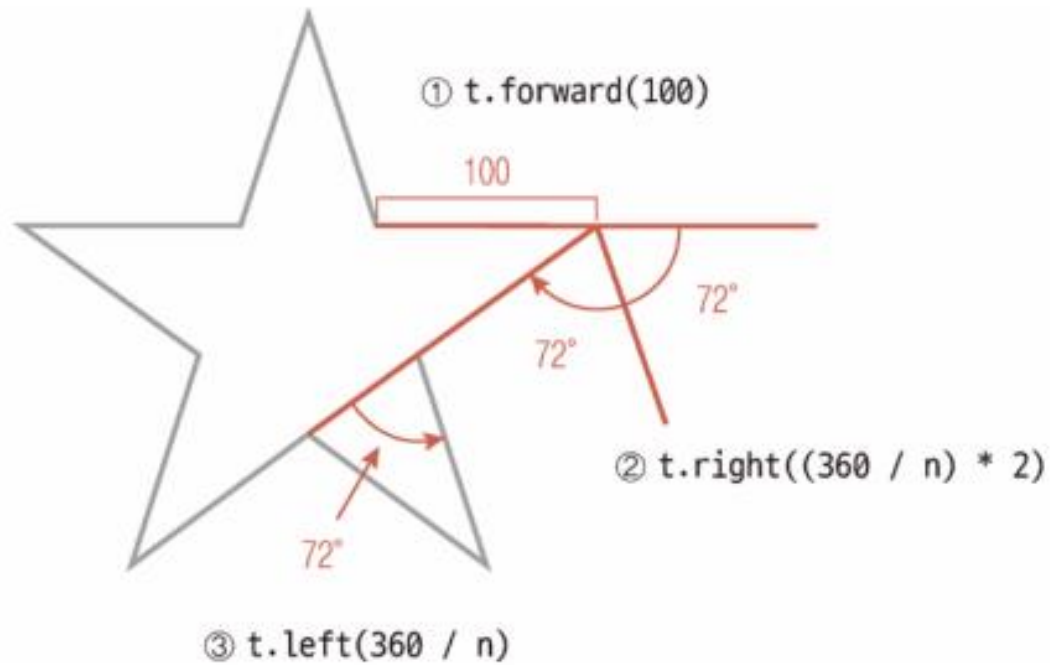
중첩 for문을 이용한 구구단 만들기

2에서 9까지
증가 후 종료
(안쪽 for 문
: k 변수)

| 1 증가 | | | | 1 증가 | | | | 1 증가 | | | | 1 증가 | | | |
|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|
| 2 × 1 = 2 | 3 × 1 = 3 | 4 × 1 = 4 | 9 × 1 = 9 | 2 × 2 = 4 | 3 × 2 = 6 | 4 × 2 = 8 | 9 × 2 = 18 | 2 × 3 = 6 | 3 × 3 = 9 | 4 × 3 = 12 | 9 × 3 = 27 | 2 × 4 = 8 | 3 × 4 = 12 | 4 × 4 = 16 | 9 × 4 = 36 |
| 2 × 5 = 10 | 3 × 5 = 15 | 4 × 5 = 20 | 9 × 5 = 45 | 2 × 6 = 12 | 3 × 6 = 18 | 4 × 6 = 24 | 9 × 6 = 54 | 2 × 7 = 14 | 3 × 7 = 21 | 4 × 7 = 28 | 9 × 7 = 63 | 2 × 8 = 16 | 3 × 8 = 24 | 4 × 8 = 32 | 9 × 8 = 72 |
| 2 × 9 = 18 | 3 × 9 = 27 | 4 × 9 = 36 | 9 × 9 = 81 | | | | | | | | | | | | |

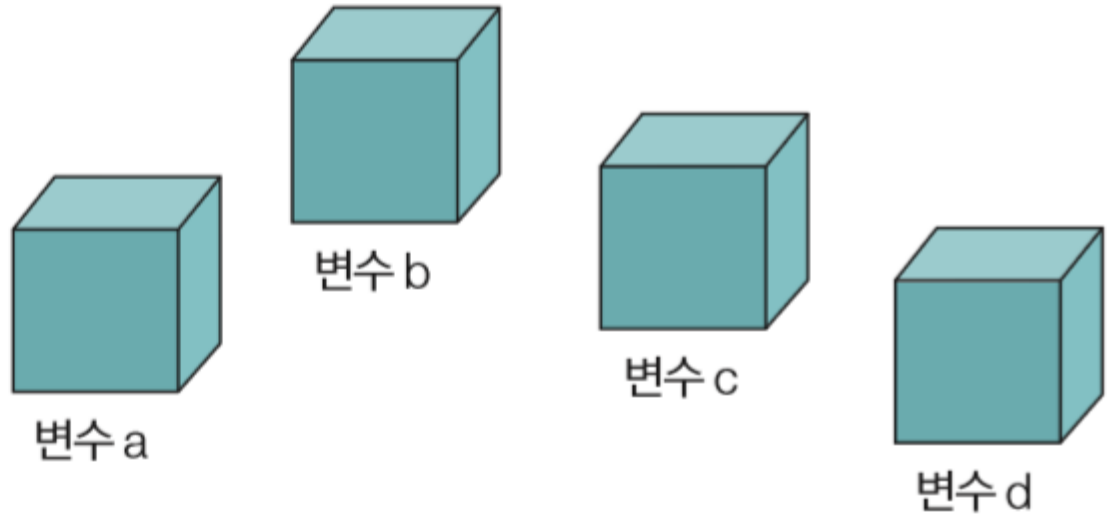
1회에서 9까지 증가 후 종료(바깥 for 문 : i 변수)

터틀 그래픽스 모듈로 별 그리기



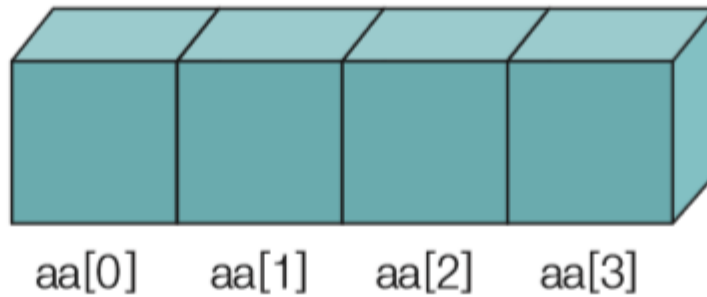
리스트 (list)

하나씩
변수로 사용



리스트로
정의해 사용

리스트 aa



리스트 선언하고 요소에 접근하기

- **요소** (element)
 - 리스트의 대괄호 내부에 넣는 자료

```
[요소, 요소, 요소...]
```

```
>>> [1, 2, 3, 4]                                # 숫자만으로 구성된 리스트
[1, 2, 3, 4]
>>> ["안", "녕", "하", "세", "요"]              # 문자열만으로 구성된 리스트
['안', '녕', '하', '세', '요']
>>> [273, 32, 103, "문자열", True, False]        # 여러 자료형으로 구성된 리스트
[273, 32, 103, '문자열', True, False]
```

리스트 선언하고 요소에 접근하기

- 리스트 내부의 요소 각각 사용하려면 리스트 이름 바로 뒤에 대괄호 입력 후 자료의 위치 나타내는 숫자 입력

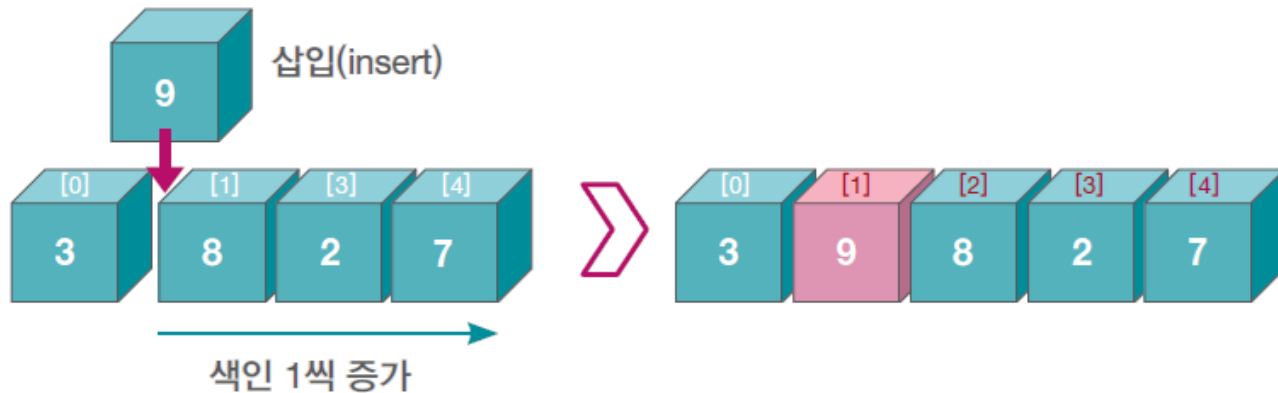
```
list_a = [273, 32, 103, "문자열", True, False]
```

| | | | | | | |
|--------|-----|-----|-----|-----|------|-------|
| list_a | 273 | 32 | 103 | 문자열 | True | False |
| | [0] | [1] | [2] | [3] | [4] | [5] |

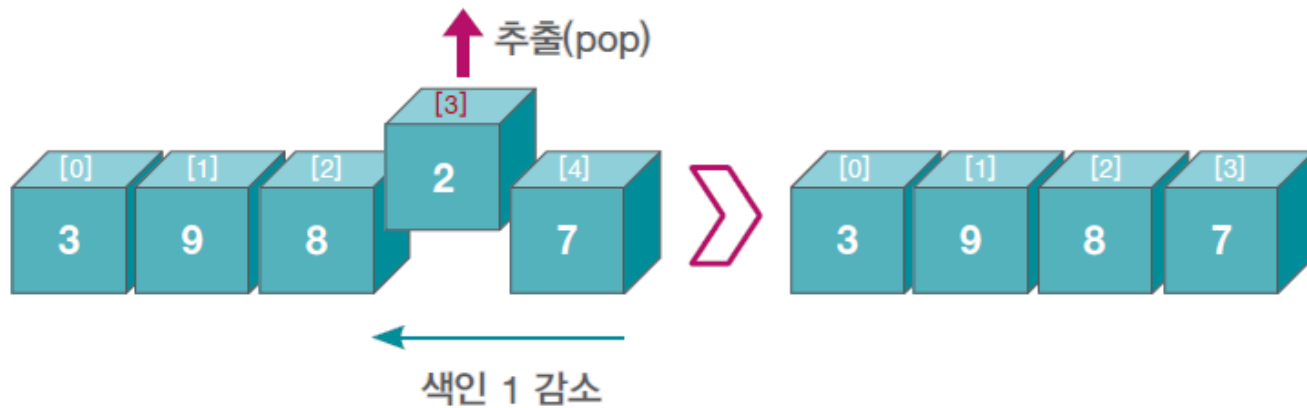
- 인덱스 (index)
 - 대괄호 안에 들어간 숫자

append(), insert()

추가(append)

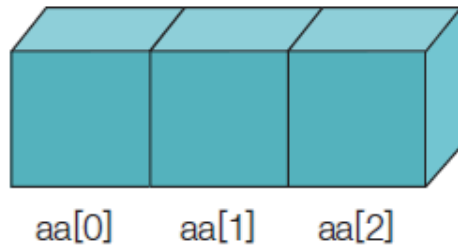


Remove(), pop()

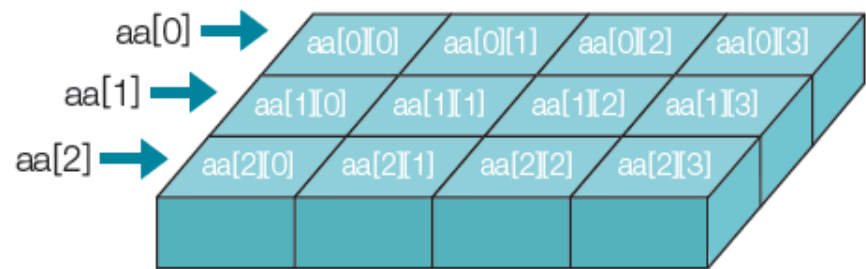


1차원 리스트와 2차원 리스트

aa = [10, 20, 30]



aa = [[1, 2, 3, 4] ,
[5, 6, 7, 8] ,
[9, 10, 11, 12]]



전체 리스트 이름 : aa

딕셔너리

```
{  
    키↓   값↓  
    "키A": 10,      # 문자열을 키로 사용하기  
    "키B": 20,  
    "키C": 30,  
    1:      40,      # 숫자를 키로 사용하기  
    False: 50        # 불을 키로 사용하기  
}
```

| 자료형 | 의미 | 가리키는 위치 | 선언 형식 |
|------|-----------------|---------|---------|
| 리스트 | 인덱스를 기반으로 값을 저장 | 인덱스 | 변수 = [] |
| 딕셔너리 | 키를 기반으로 값을 저장 | 키 | 변수 = {} |

함수

파이썬 스크립트

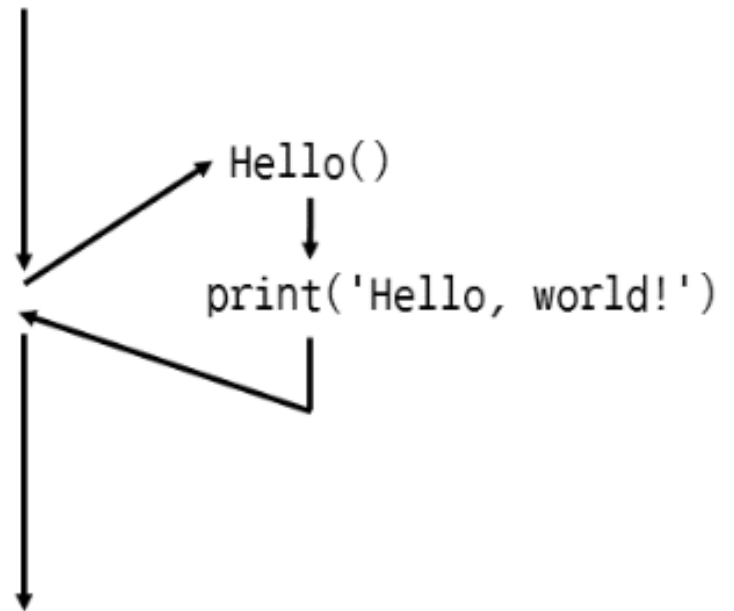
①

```
def hello(): ③  
    print('Hello, world!') ④
```

⑤

```
hello() ②
```

⑥



함수(매개변수)

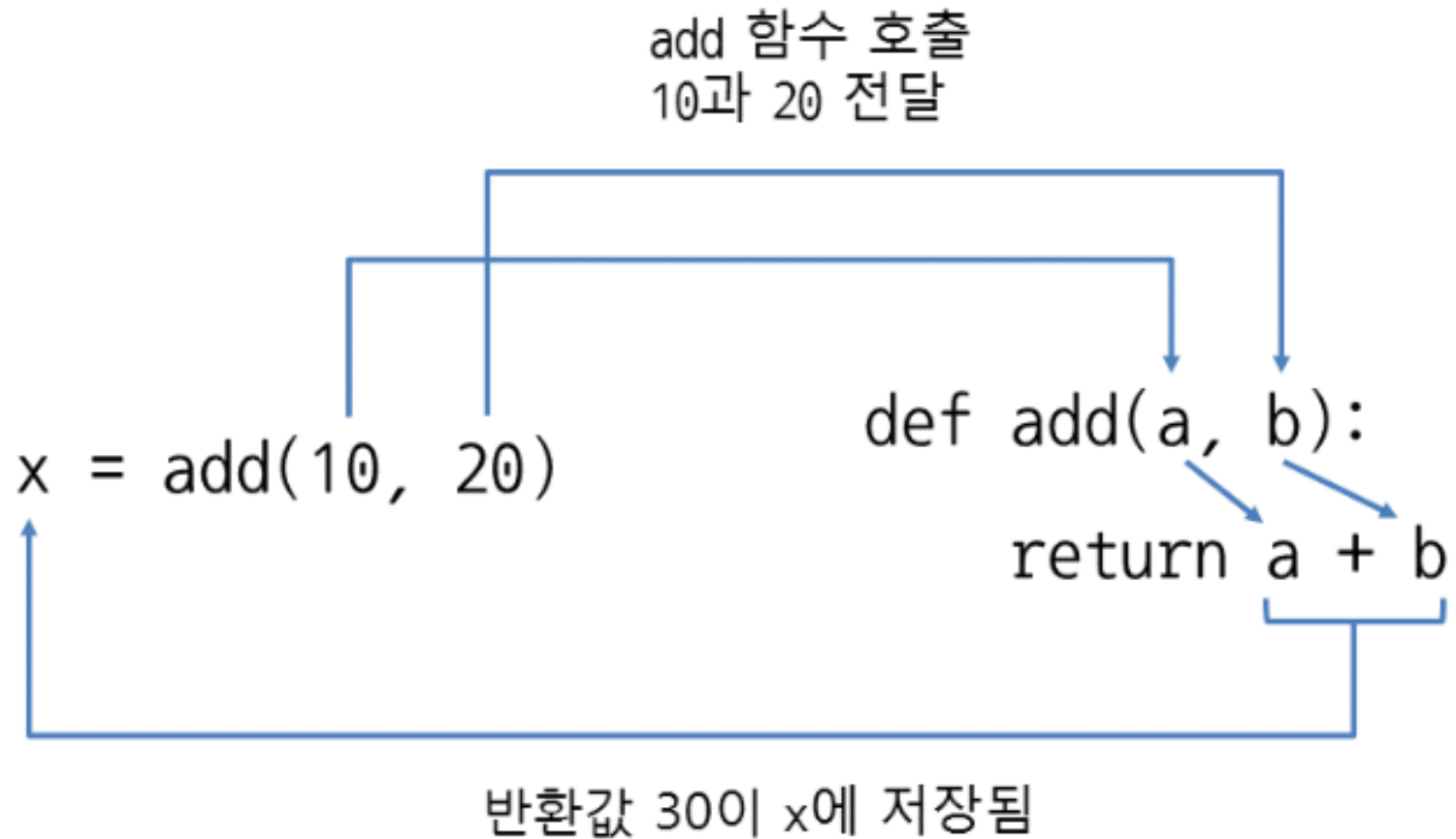
add 함수 호출
10과 20 전달

add(10, 20)

```
def add(a, b):  
    print(a + b) 30 출력
```

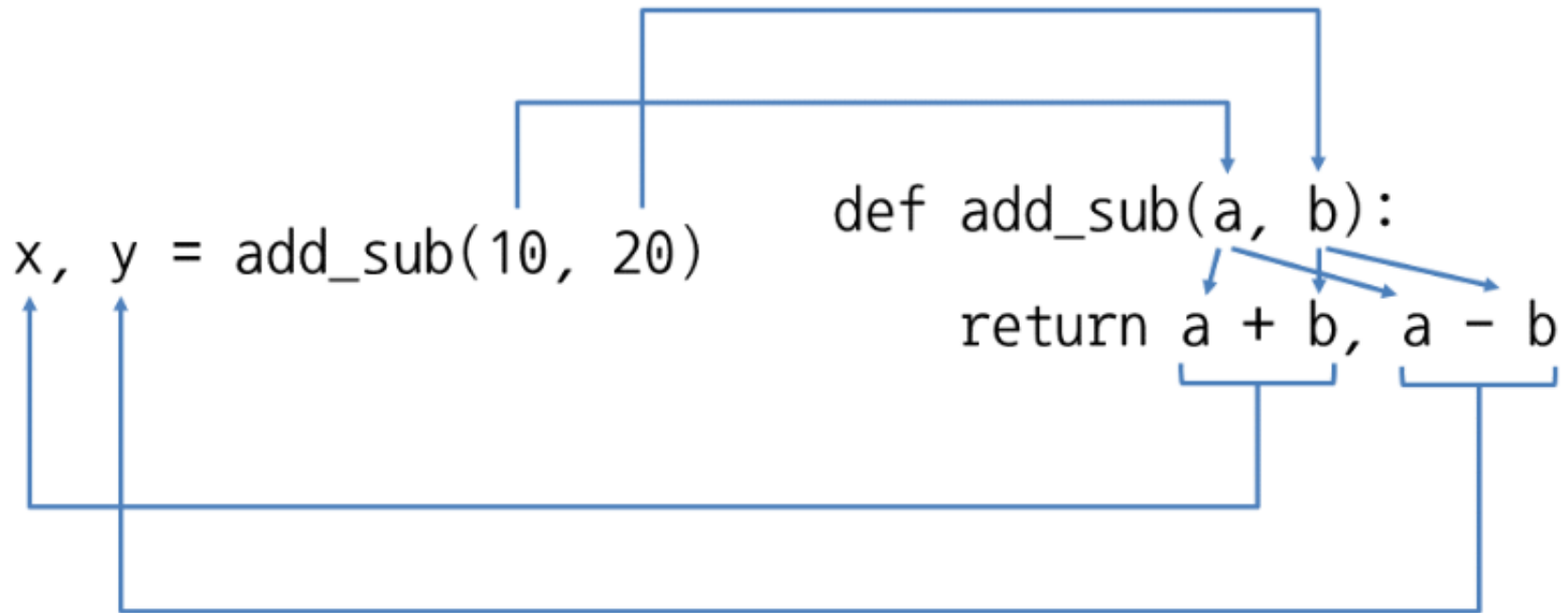
```
graph LR; Call[add(10, 20)] --> ParamA[a]; Call --> ParamB[b]; ParamA --> Def[def add(a, b)]; ParamB --> Def; Def --> Body[print(a + b)]; Body --> Result[30 출력];
```


함수(반환값 1개)



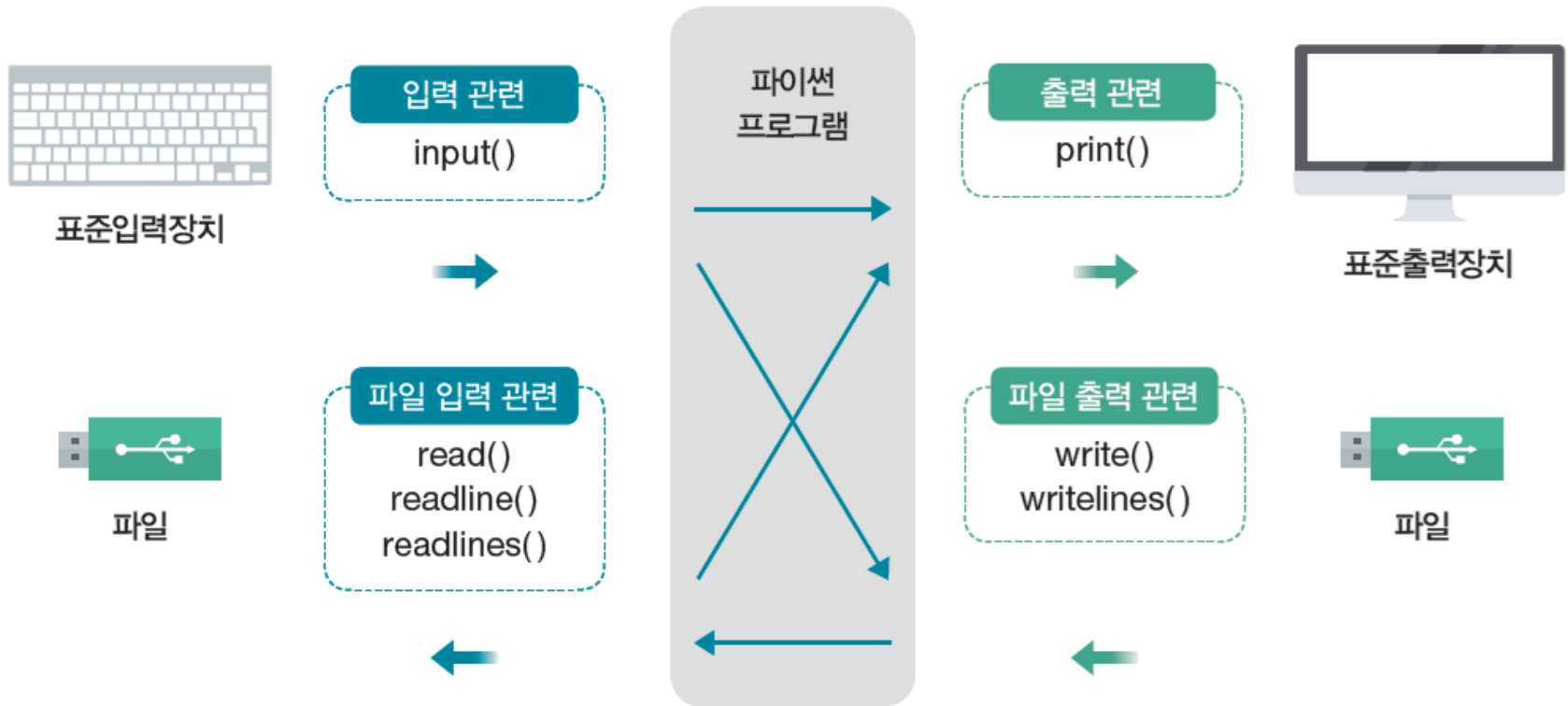
함수(반환값 여러 개)

add_sub 함수 호출
10과 20 전달



반환값 30은 x, -10은 y에 저장됨

파일 입출력 과정



파일 입출력 과정

1단계



파일 열기

2단계



파일 읽기 및
파일 쓰기 작업

3단계



파일 닫기

파일에 문자열 쓰기

`file = open()`

파일 열기



`file.write()`

파일 쓰기



`file.close()`

파일 닫기

```
file = open()
```

파일 열기



```
file.read()
```

파일 읽기

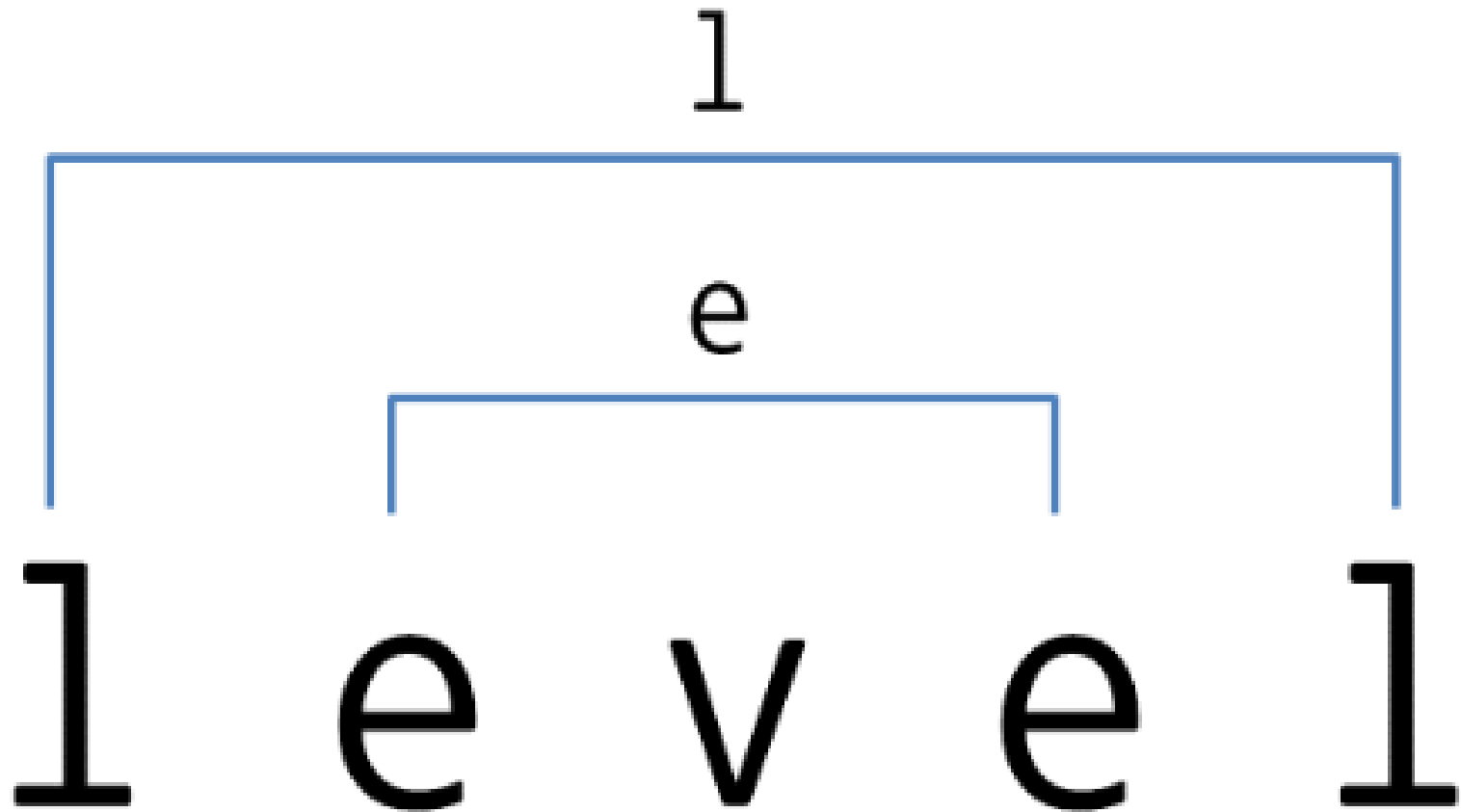


```
file.close()
```

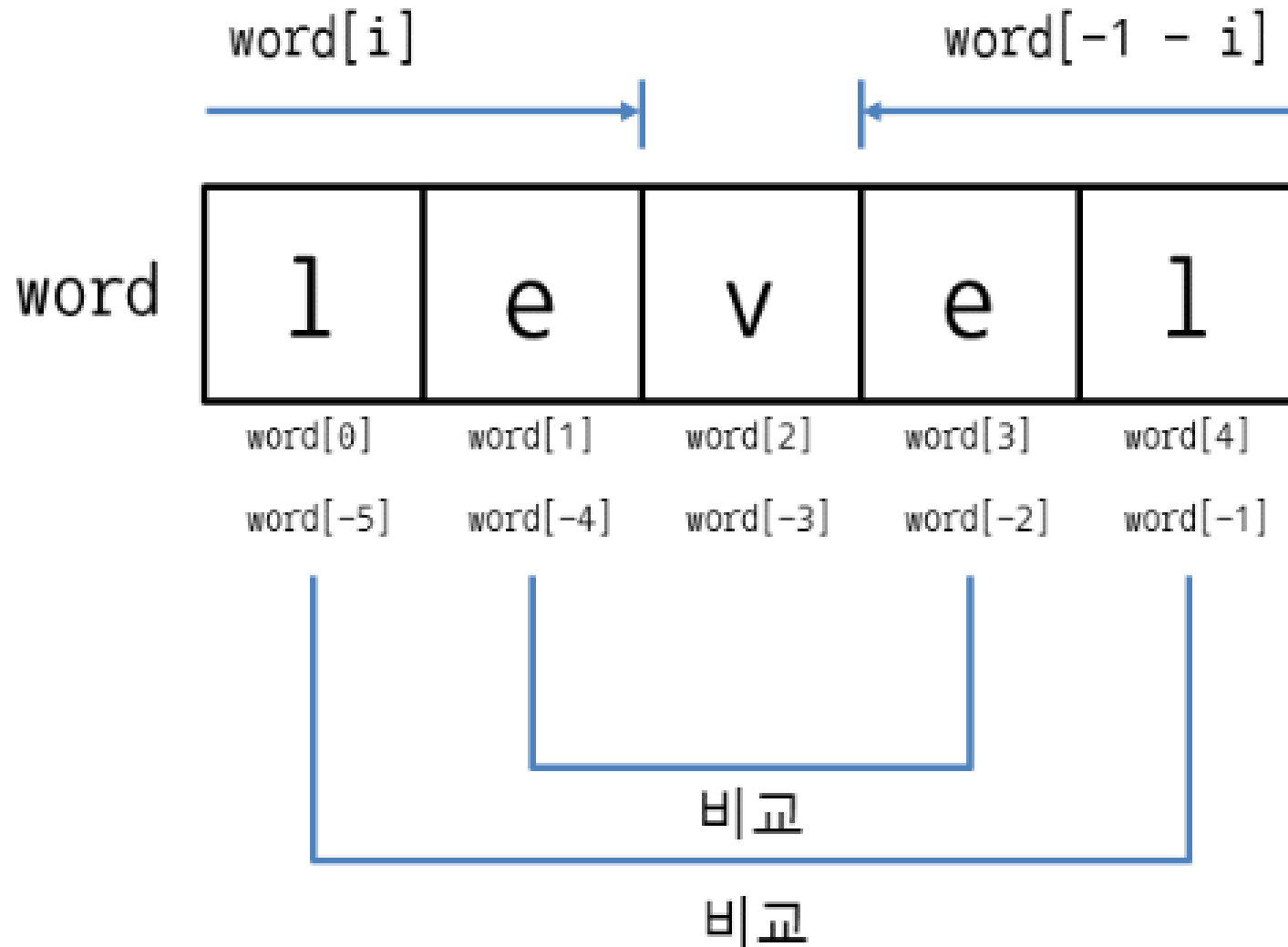
파일 닫기

회문(palindrome)

1
1 e 1
e v e
1 e v e 1



회문 판별하기



지역 변수 / 전역 변수

❶ 지역 변수의 생존 범위

함수 1

$a = 10$

a가 뭔지 함수 1에서 안다.

함수 2

a가 뭔지 함수 2에서 모른다.

❷ 전역 변수의 생존 범위

함수 1

b가 뭔지 함수 1에서 안다.

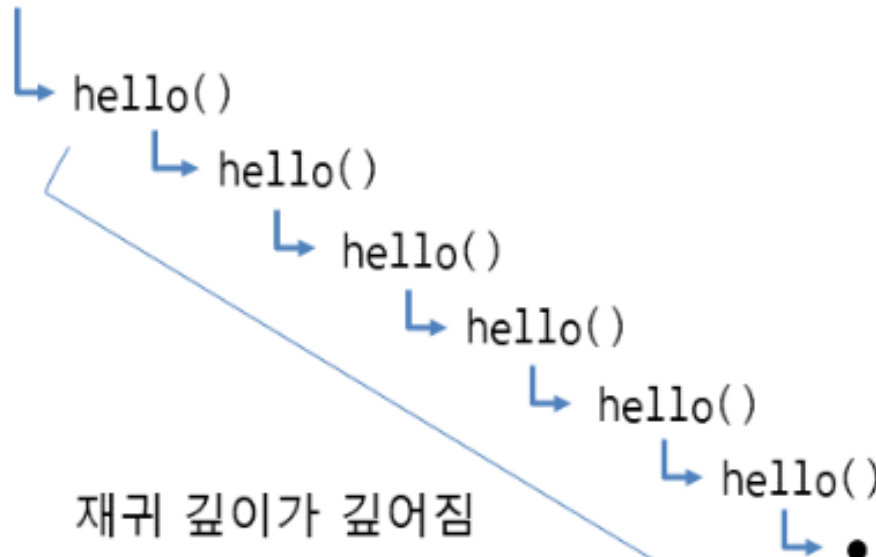
함수 2

b가 뭔지 함수 2에서 안다.

$b = 20$

재귀 함수

```
def hello():  
    print('Hello, world!')  
    hello()
```



재귀 깊이가 깊어짐

최대 재귀 깊이를 초과하면
RecursionError가 발생함

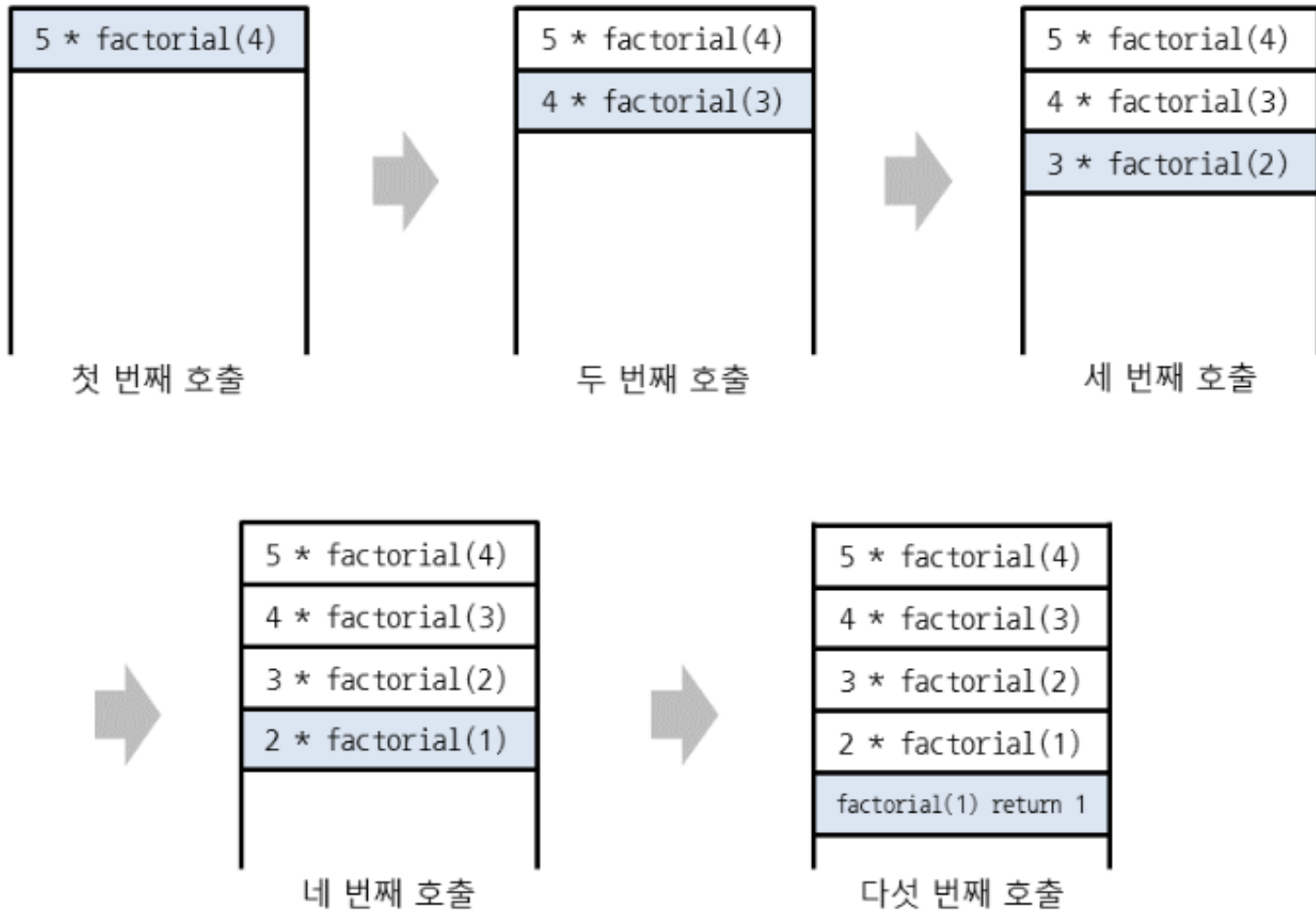
재귀 함수 – 종료 조건 지정

5
↓

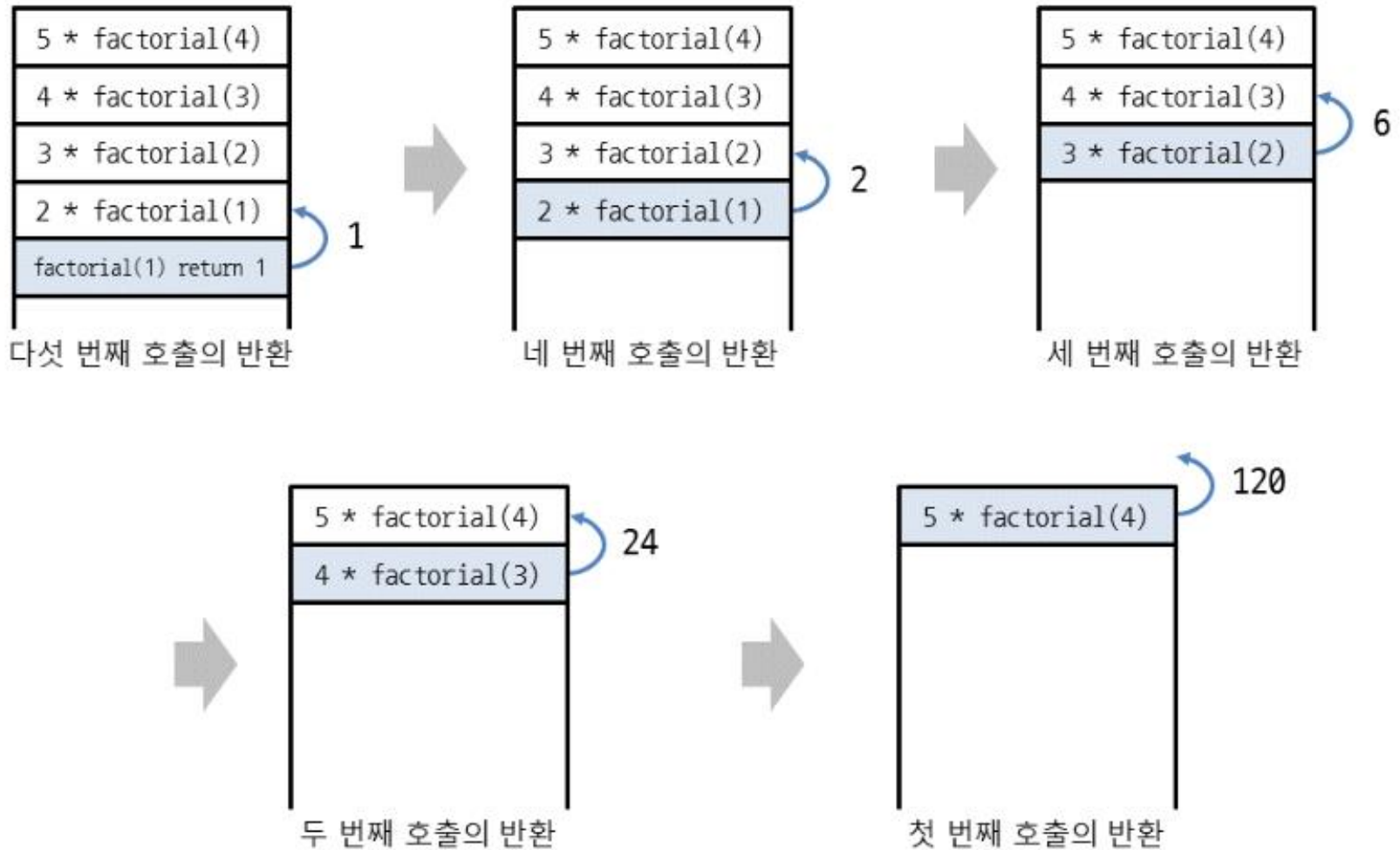
```
def hello(count):  
    if count == 0:  
        return  
  
    print('Hello, world!', count)  
  
    count -= 1  
    hello(count)
```

└─> hello(4)
 └─> hello(3)
 └─> hello(2)
 └─> hello(1)
 └─> hello(0) 종료 조건을 만족하므로 재귀호출을 끝냄

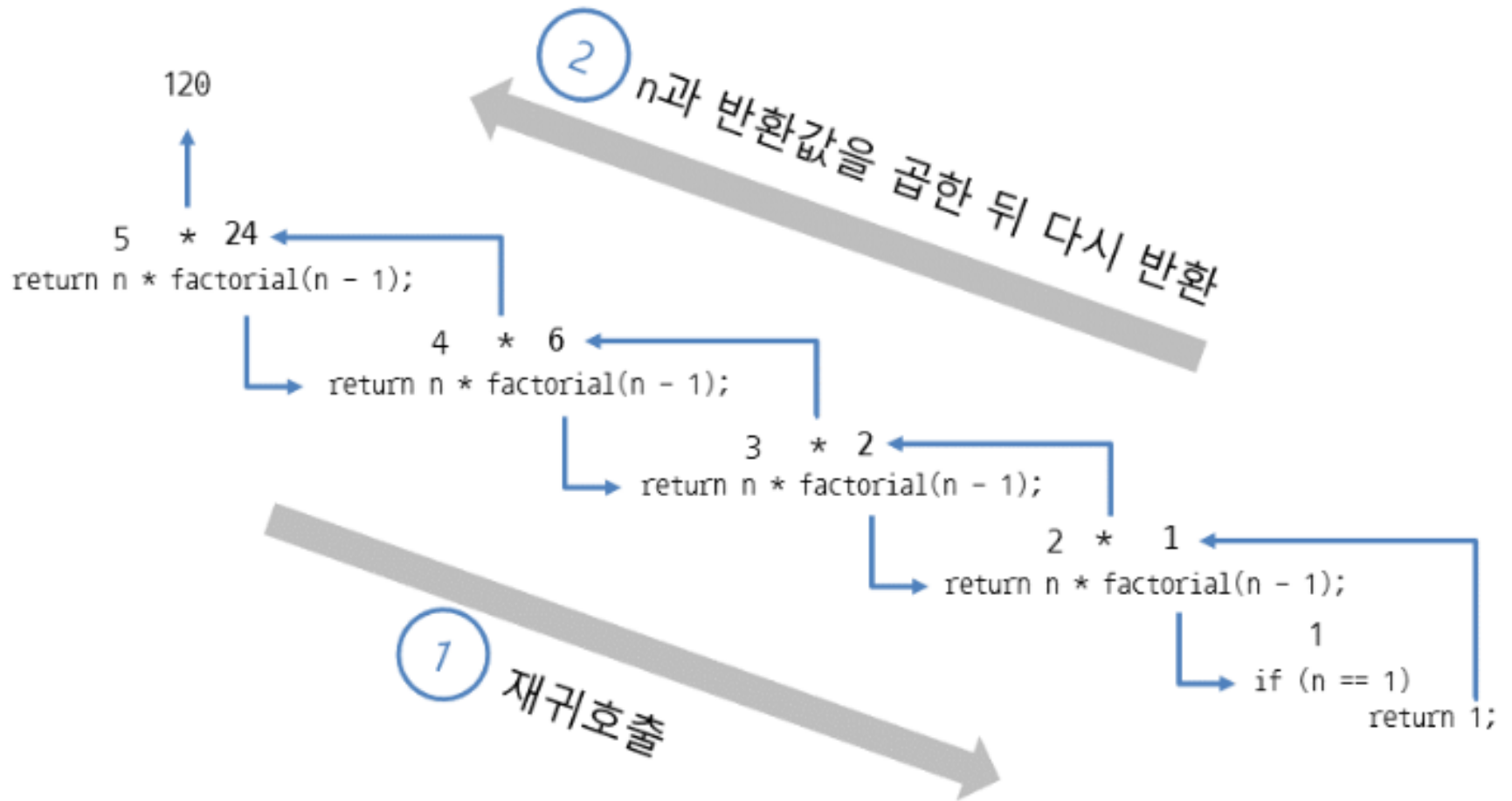
재귀 함수 – 팩토리얼 구하기



재귀 함수 – 팩토리얼 구하기



재귀 함수 – 팩토리얼 구하기



예외 처리

0

try:

$x = \text{int}(\text{input}(\text{'나눌 숫자를 입력하세요: '}))$

$y = 10 / x$
 $\text{print}(y)$

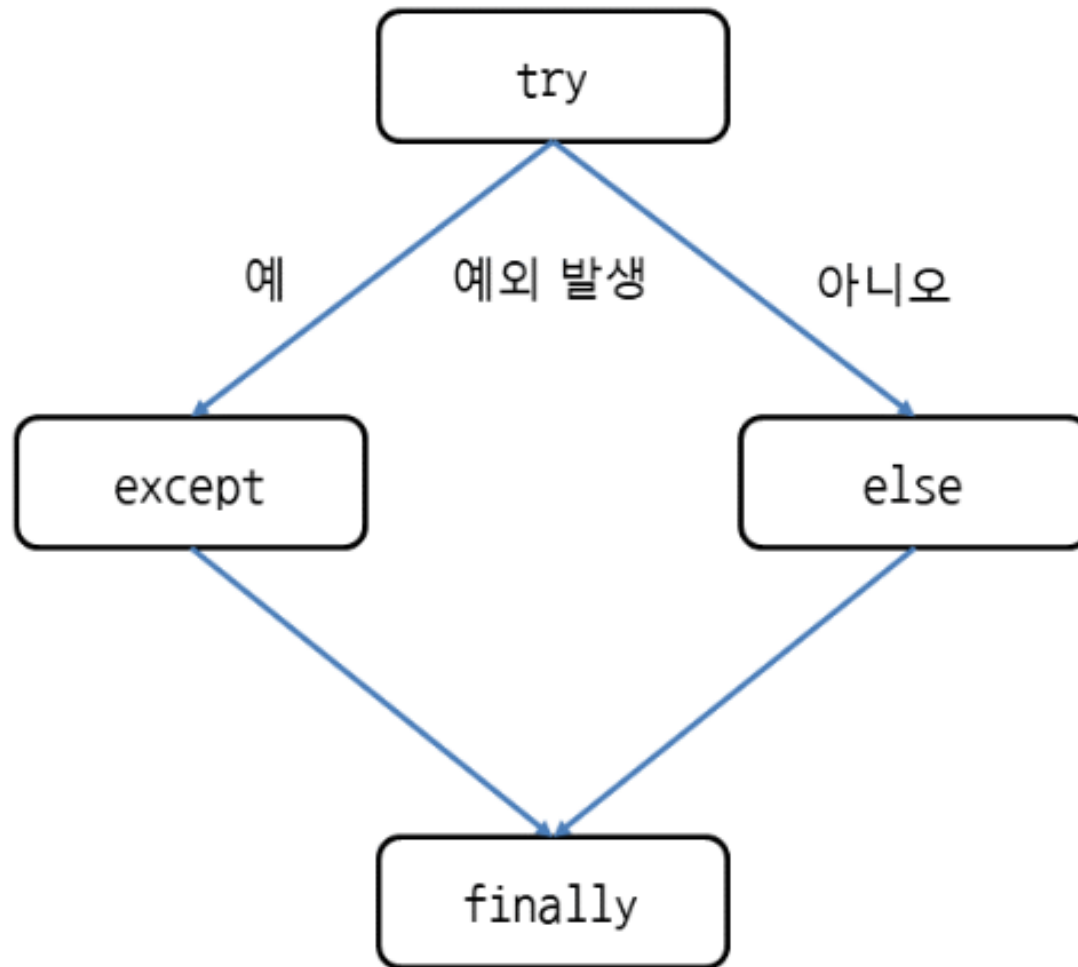
except:

$\text{print}(\text{'예외가 발생했습니다.'})$

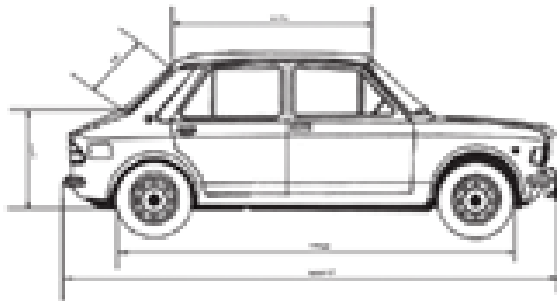
실행되지 않음

예외가 발생하면 코드 실행을 중단하고
바로 except로 가서 코드 실행

예외 처리



클래스



class 자동차 :

자동차의 속성

색상

속도

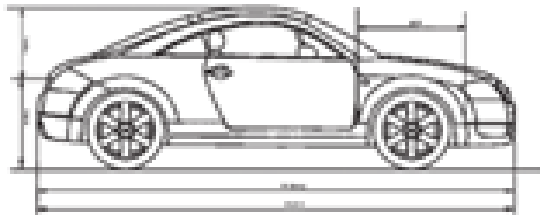
자동차의 기능

속도 올리기()

속도 내리기()

클래스와 인스턴스

자동차 설계도(클래스)



여러 번
찍어 내기



자동차(인스턴스)



생성자

```
class block_factory():
```

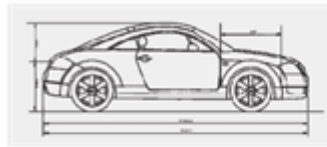
```
    def __init__(self, company, color, shape):
```

```
newblock = block_factory("Gole", "blue", "long")
```

The diagram illustrates the argument passing process. A blue arrow originates from the 'self' parameter in the `__init__` method definition and points to the `newblock` variable in the function call. Three additional blue arrows point from the string arguments in the function call to the corresponding parameters: from `"Gole"` to `company`, from `"blue"` to `color`, and from `"long"` to `shape`.

인스턴스 변수

자동차 설계도(클래스)



색상
속도



자동차(인스턴스)



색상

속도



색상

속도

```
class Car :  
    color = ""  
    speed = 0
```



```
myCar1 = Car()
```

myCar1.color

myCar1.speed

```
myCar2 = Car()
```

myCar2.color

myCar2.speed

클래스 변수

자동차 설계도(클래스)



색상

속도

수량



공유

자동차(인스턴스)



색상

속도

수량



색상

속도

수량

```
class Car :  
    color = ""  
    speed = 0
```

count



공유

```
myCar1 = Car()
```

myCar1.color

myCar1.speed

Car.count

```
myCar2 = Car()
```

myCar2.color

myCar2.speed

myCar2.count

클래스의 상속

class 자동차 :

필드 - 색상, 속도

메서드 - 속도 올리기()

속도 내리기()

자동차 클래스(공통 내용)



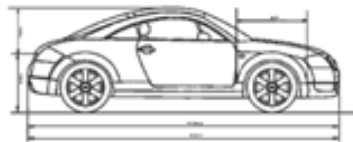
상속



상속



승용차 클래스



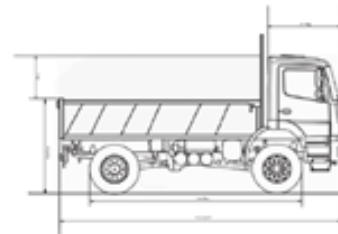
class 승용차 : 자동차 상속

필드 - 자동차 필드, 좌석 수

메서드 - 자동차 메서드

좌석 수 알아보기()

트럭 클래스



class 트럭 : 자동차 상속

필드 - 자동차 필드, 적재량

메서드 - 자동차 메서드

적재량 알아보기()

메서드 오버라이딩

class 자동차:

메서드 - 속도 올리기()

자동차 클래스(공통 내용)



⚙️ 속도 올리기()

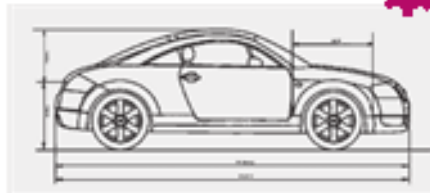
상속



상속

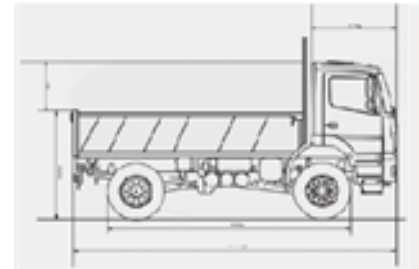


승용차 클래스



⚙️ 속도 올리기()

트럭 클래스



class 승용차(자동차):

메서드 - 속도 올리기() 재정의

class 트럭(자동차):

메서드 -