

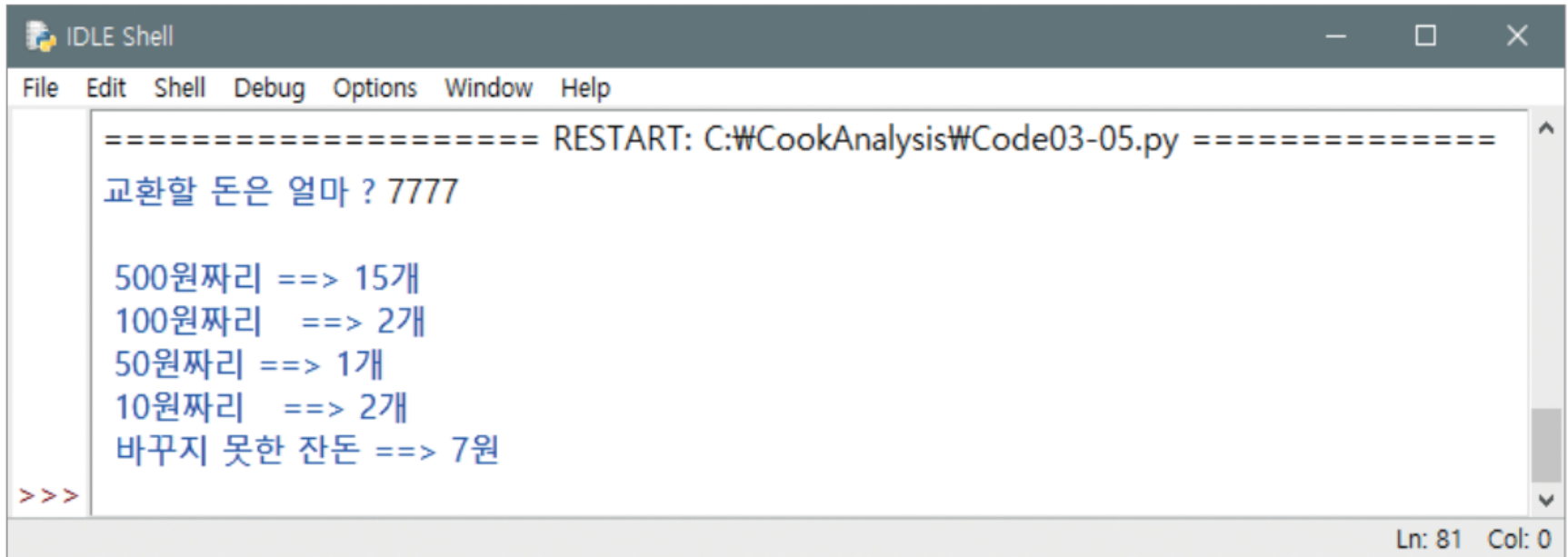
# 학습 목표

- `print()` 함수 서식에 맞추어 출력하는 방법을 익힌다.
- 정수, 실수, 문자열 등 기본 데이터형을 알아본다.
- 산술 연산자 및 관계·논리 연산자의 사용법을 익힌다.
- 조건문의 대표 격인 기본 `if` 문, 중첩 `if` 문의 형식과 사용법을 익힌다.
- 조건문을 다양하게 활용하는 방법을 익힌다.
- 기본 `for` 문 및 중첩 `for` 문의 형식과 사용법을 익힌다.
- `while` 문, `break` 문, `continue` 문의 사용법을 익힌다.

## Section 01 이 장에서 만들 프로그램

### ■ [프로그램 1] 동전 교환

- 입력된 액수를 500원, 100원, 50원, 10원짜리 동전으로 교환하는 프로그램



```
===== RESTART: C:\CookAnalysis\Code03-05.py =====
교환할 돈은 얼마 ? 7777

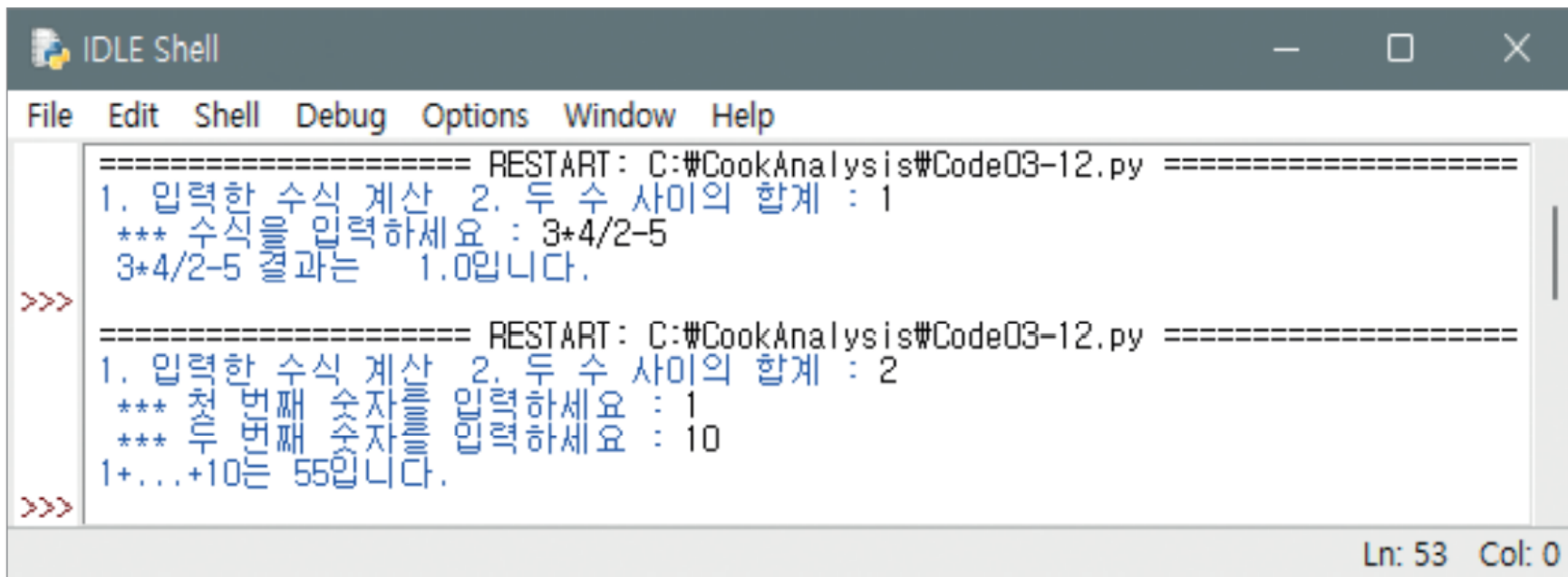
500원짜리 ==> 15개
100원짜리  ==> 2개
50원짜리   ==> 1개
10원짜리   ==> 2개
바꾸지 못한 잔돈 ==> 7원
>>>
```

Ln: 81 Col: 0

## Section 01 이 장에서 만들 프로그램

### ■ [프로그램 2] 종합 계산기

- 입력한 수식대로 계산하는 기능과 두 숫자 사이의 수를 모두 더하는 기능을 가진 종합 계산기 프로그램



```
===== RESTART: C:\#CookAnalysis\#Code03-12.py =====
1. 입력한 수식 계산 2. 두 수 사이의 합계 : 1
*** 수식을 입력하세요 : 3*4/2-5
3*4/2-5 결과는 1.0입니다.
>>>

===== RESTART: C:\#CookAnalysis\#Code03-12.py =====
1. 입력한 수식 계산 2. 두 수 사이의 합계 : 2
*** 첫 번째 숫자를 입력하세요 : 1
*** 두 번째 숫자를 입력하세요 : 10
1+...+10는 55입니다.
>>>
```

Ln: 53 Col: 0

## Section 02 print() 함수와 출력

### ■ print() 함수와 서식

- 다음 구문은 큰따옴표 안의 내용을 화면에 출력하므로 결과는 “안녕하세요?”

```
print("안녕하세요?")
```

- ❶의 결과로 나온 100은 숫자 100이 아닌 문자 '100'
- ❷의 결과로 나온 100은 숫자 100을 의미

```
❶ print("100")
```

```
❷ print("%d" % 100)
```

## Section 02 print() 함수와 출력

### ■ print() 함수와 서식

❸ `print("100 + 100")`

❹ `print("%d" % (100 + 100))`

- ❸은 100+100이 출력
- ❹는 숫자 100과 숫자 100을 더한 결과인 숫자 200을 출력
- 서식은 앞에 퍼센트(%)가 붙는데, ❹에서 %d는 정수를 의미
- 서식의 개수와 따옴표 뒤에 나오는 숫자(또는 문자)의 개수는 같아야 함

## Section 02 print() 함수와 출력

### ■ print() 함수와 서식

- 서식과 숫자의 대응이 잘못된 예시

❶ `print("%d" % (100, 200))`

❷ `print("%d %d" % (100))`

- ❶은 %d가 1개밖에 없는데 숫자가 2개이고, ❷은 %d가 2개인데 숫자는 1개라 서로 짝이 맞지 않음
- ❷은 단순히 %d를 하나 삭제하면 되지만 ❶은 숫자 2개를 출력하려면 %d가 2개 필요

## Section 02 print() 함수와 출력

- **print() 함수와 서식**
  - 서식과 숫자의 대응이 올바르게 된 예시

The diagram illustrates the mapping of format specifiers to arguments in the code `print( "%d %d" % ( 100 , 200 ) )`. A purple circle highlights the percent sign (`%`) in the format string. Two teal arrows originate from this circle: one points to the first `%d` in the format string, and the other points to the first argument `100` in the tuple. A second teal arrow points from the second `%d` in the format string to the second argument `200` in the tuple, showing the correct correspondence between the format specifiers and the values provided.

```
print( "%d %d" % ( 100 , 200 ) )
```

그림 3-1 서식과 숫자의 대응

## Section 02 print() 함수와 출력

### ■ print() 함수와 서식

- 서식과 숫자의 불일치 상황

```
print("%d / %d = %d" % (100, 200, 0.5))
```

- 세 번째 숫자인 0.5는 소수점이 있는 실수인데, 서식에 정수형(%d)을 사용했기 때문

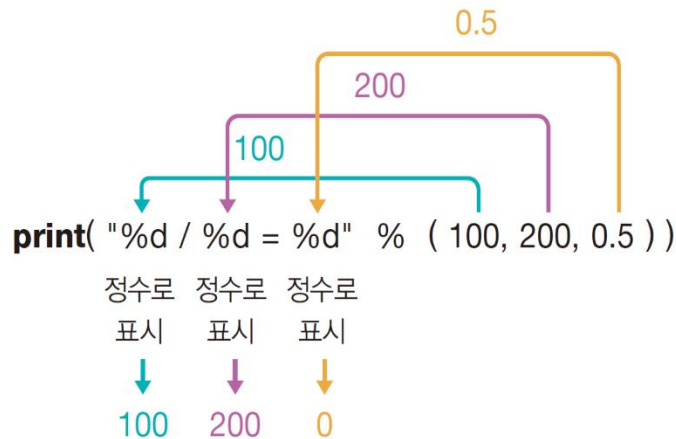


그림 3-2 서식과 숫자의 불일치 상황



## Section 02 print() 함수와 출력

### ■ print() 함수와 서식

#### ■ 출력값에 따른 서식의 종류

표 3-1 print() 함수에서 사용할 수 있는 서식

서식	값의 예	설명
%d, %x, %o	10, 100, 1234	정수(10진수, 16진수, 8진수)
%f	0.5, 1.0, 3.14	실수(소수점이 붙은 수)
%c	"b", "한"	한 글자
%s	"안녕", "abcdefg", "a"	두 글자 이상인 문자열

#### ■ 값 0.5 출력 코드 (%5.1f 는 다음 절에서 다루도록 함)

```
print("%d / %d = %5.1f" % (100, 200, 0.5))
```

## Section 02 print() 함수와 출력

### ■ 서식에 따른 출력

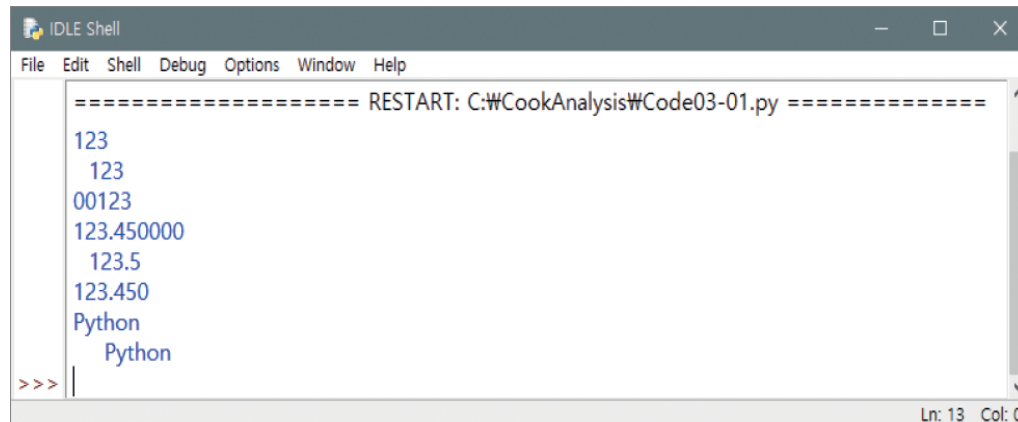
Code03-01.py

```
01 print("%d" % 123)
02 print("%5d" % 123)
03 print("%05d" % 123)
04
05 print("%f" % 123.45)
06 print("%7.1f" % 123.45)
07 print("%7.3f" % 123.45)
08
09 print("%s" % "Python")
10 print("%10s" % "Python")
```

정수형 데이터 출력

실수형 데이터의 서식 지정

문자형 데이터의 서식 지정



```
===== RESTART: C:\CookAnalysis\Code03-01.py =====
123
123
00123
123.450000
123.5
123.450
Python
Python
>>>
```

## Section 02 print() 함수와 출력

### ■ 서식에 따른 출력

#### ■ 정수형 데이터의 서식 지정

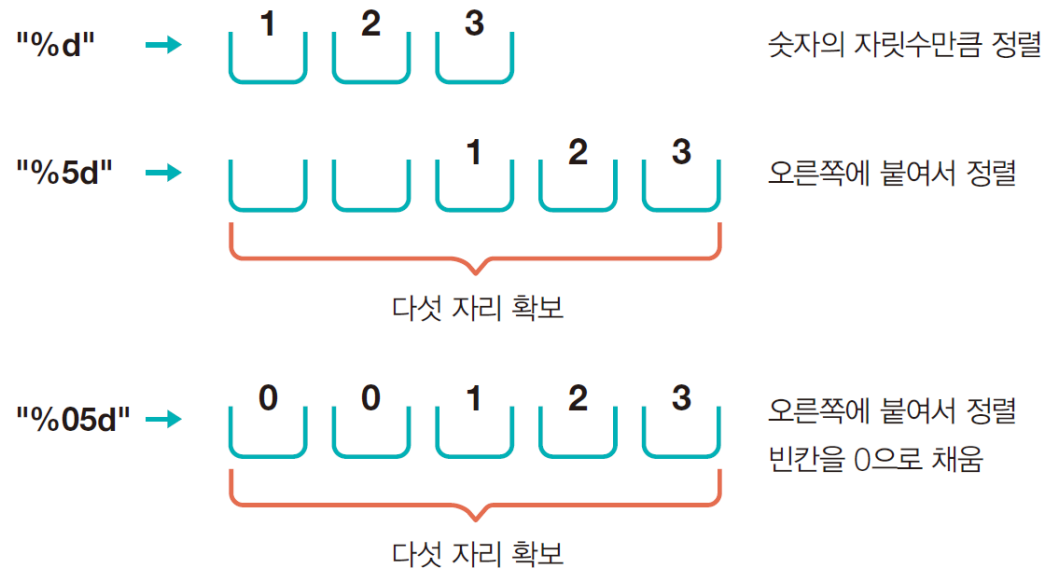


그림 3-3 정수형 데이터의 서식 지정

## Section 02 print() 함수와 출력

### ■ 서식에 따른 출력

#### ■ 실수형 데이터의 서식 지정

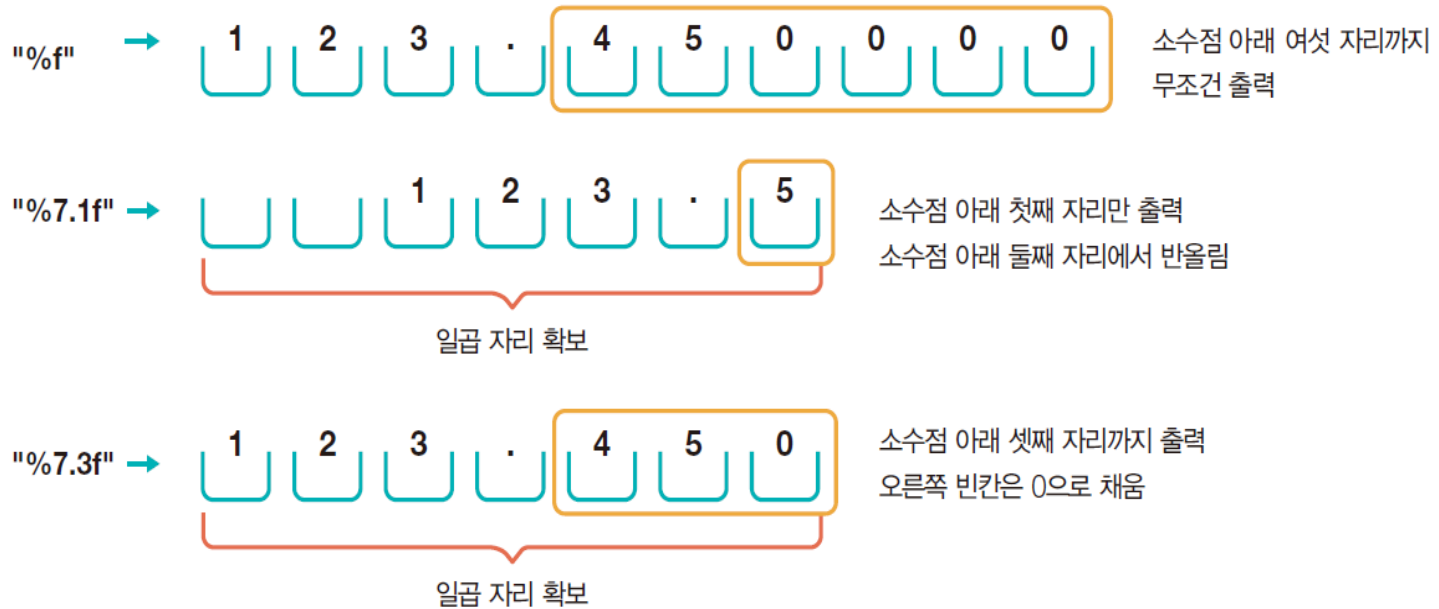


그림 3-4 실수형 데이터의 서식 지정

## Section 02 print() 함수와 출력

- 서식에 따른 출력
  - 문자열 데이터의 서식 지정

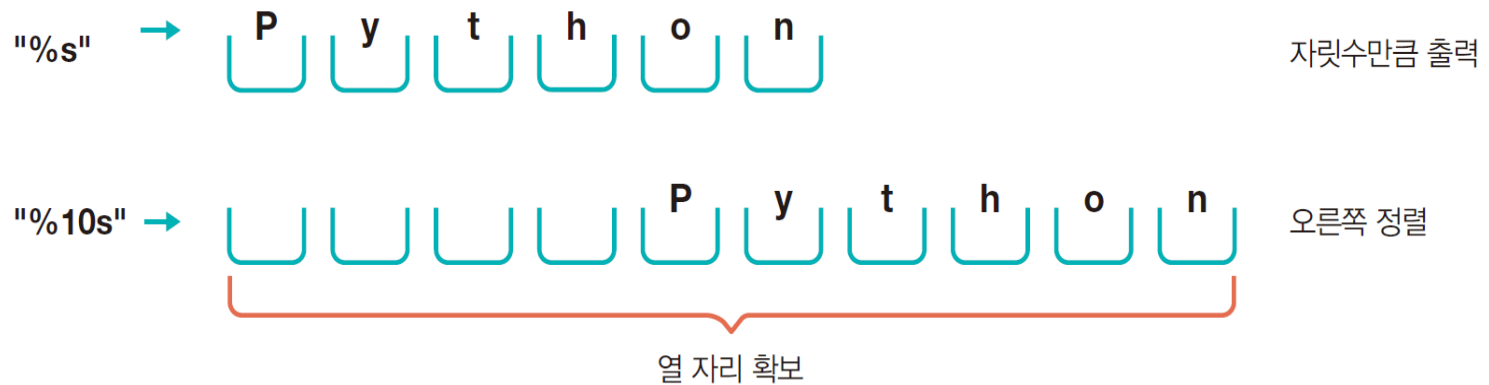


그림 3-5 문자열 데이터의 서식 지정

## Section 02 print() 함수와 출력

### ■ 서식에 따른 출력

- format 함수와 중괄호를 함께 사용한 서식 지정

```
print("%d %5d %05d" % (123, 123, 123))  
print("{0:d} {1:5d} {2:05d}".format(123, 123, 123))
```

- 두 코드의 실행 결과는 동일함
- 중괄호({}) 안의 0, 1, 2는 format() 함수 안의 값 중에서 0번째, 1번째, 2번째에 대응한다는 의미

## Section 02 print() 함수와 출력

- 서식에 따른 출력
  - format 함수의 사용

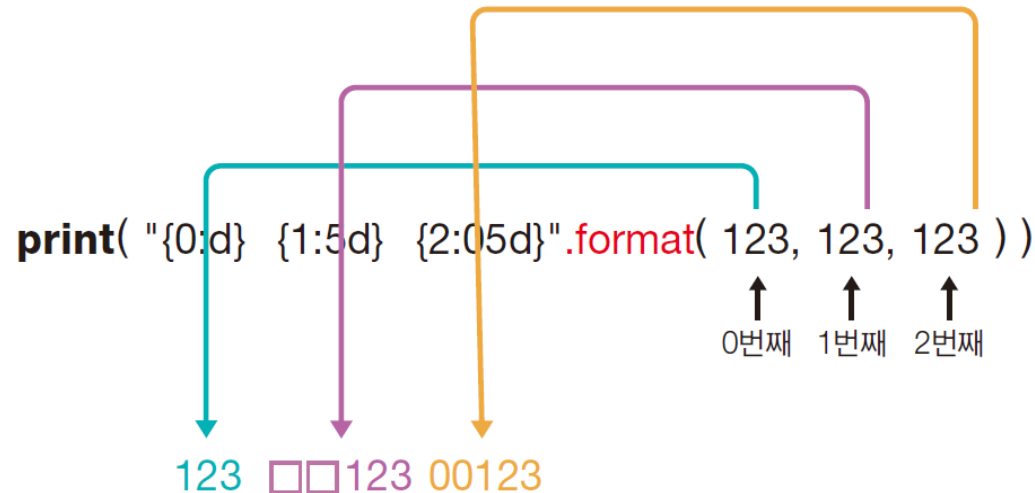


그림 3-6 format() 함수의 사용

## Section 02 print() 함수와 출력

### ■ 서식에 따른 출력

- format 함수는 출력하는 순서를 지정 가능
- 다음 코드는 300, 200, 100 을 출력함

```
print("{2:d} {1:d} {0:d}".format(100, 200, 300))
```

- print() 함수는 내용 출력 후 한 행을 넘김
- 강제로 한 행을 넘기고 싶다면 \n을 사용

```
print("한 행입니다. 또 한 행입니다.")  
print("한 행입니다. \n또 한 행입니다.")
```



## Section 02 print() 함수와 출력

### ■ 서식에 따른 출력

- \n 와 같이 앞에 백슬래시가 있는 문자를 이스케이프 문자라고 함
- 이스케이프 문자의 종류

표 3-2 이스케이프 문자

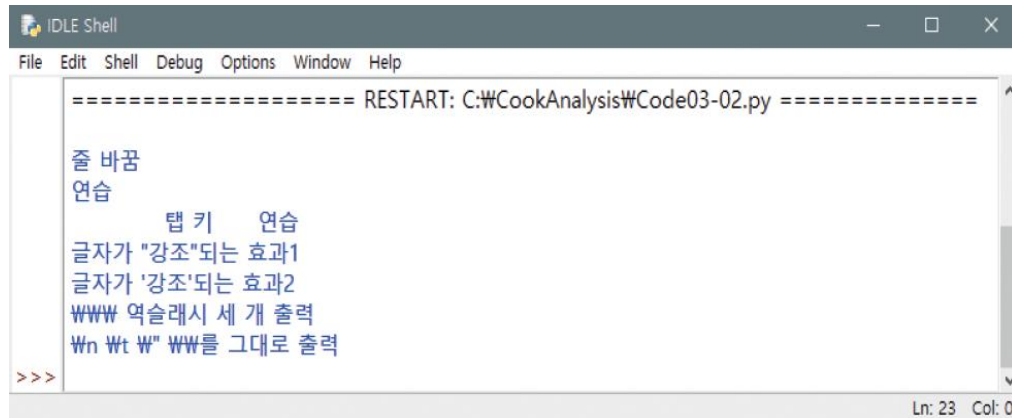
이스케이프 문자	역할	설명
\n	새로운 줄로 이동	<code>Enter</code> 를 누른 효과
\t	다음 탭으로 이동	<code>Tab</code> 을 누른 효과
\b	뒤로 한 칸 이동	<code>Backspace</code> 를 누른 효과
\\	\ 출력	
\'	' 출력	
\"	" 출력	

## Section 02 print() 함수와 출력

### ■ 서식에 따른 출력

Code03-01.py

```
01 print("\n줄바꿈\n연습 ")
02 print("\t탭키\t연습")
03 print("글자가 \"강조\"되는 효과1")
04 print("글자가 \'강조\'되는 효과2")
05 print("\\\\\\\\\\\\ 역슬래시 세 개 출력")
06 print(r"\n \t \" \\" "를 그대로 출력")
```



```
===== RESTART: C:\CookAnalysis\Code03-02.py =====

줄 바꿈
연습
    탭 키    연습
글자가 "강조"되는 효과1
글자가 '강조'되는 효과2
\\\\\\\\ 역슬래시 세 개 출력
r\n \t \" \"를 그대로 출력

>>>
```

Ln: 23 Col: 0

## Section 03 변수의 선언과 사용

### ■ 변수의 선언

- 변수는 어떠한 값을 저장하는 메모리 공간
- 가장 많이 사용하는 변수의 종류는 불형(Boolean), 정수형, 실수형, 문자열 네 가지

```
boolVar = True  
intVar = 0  
floatVar = 0.0  
strVar = ""
```

## Section 03 변수의 선언과 사용

### ■ 변수의 선언

- boolVar에는 불 값인 True를, intVar에는 정수인 0을, floatVar에는 실수인 0.0을, strVar에는 문자열 ""를 초기에 대입

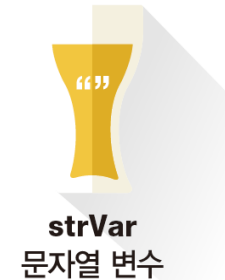


그림 3-7 변수의 종류

## Section 03 변수의 선언과 사용

### ■ 변수의 선언

- 변수명 지정의 규칙
  - 대·소문자를 구분한다. (예) myVar와 MyVar는 다른 변수
  - 문자, 숫자, 언더바(\_)를 포함할 수 있지만 숫자로 시작하면 안 된다. (예) 2Var (x)
  - 예약어는 변수명으로 쓰면 안 된다.

## Section 03 변수의 선언과 사용

### ■ 변수의 선언

- `type( )` 함수를 사용하면 변수가 `bool(불)`, `int(정수)`, `float(실수)`, `str(문자열)`형으로 생성된 것을 확인 가능

```
type(boolVar), type(intVar), type(floatVar), type(strVar)
```

#### 실행 결과

```
(<class 'bool'>, <class 'int'>, <class 'float'>, <class 'str'>)
```

## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 그릇에 음식을 담듯이 변수에 값을 담으면 (대입하면) 사용할 수 있음
- 변수에 있던 기존 값은 없어지고 새로운 값으로 변경됨

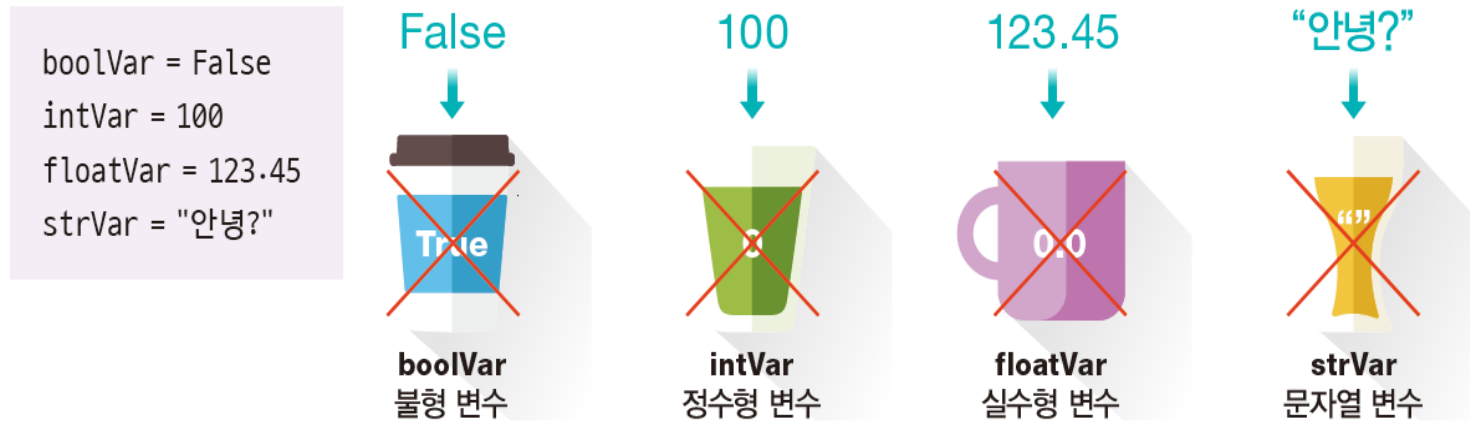


그림 3-8 변수에 값을 대입해 새로운 값으로 변경된 상태

## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 변수의 값뿐만 아니라 계산 결과를 넣을 수도 있음

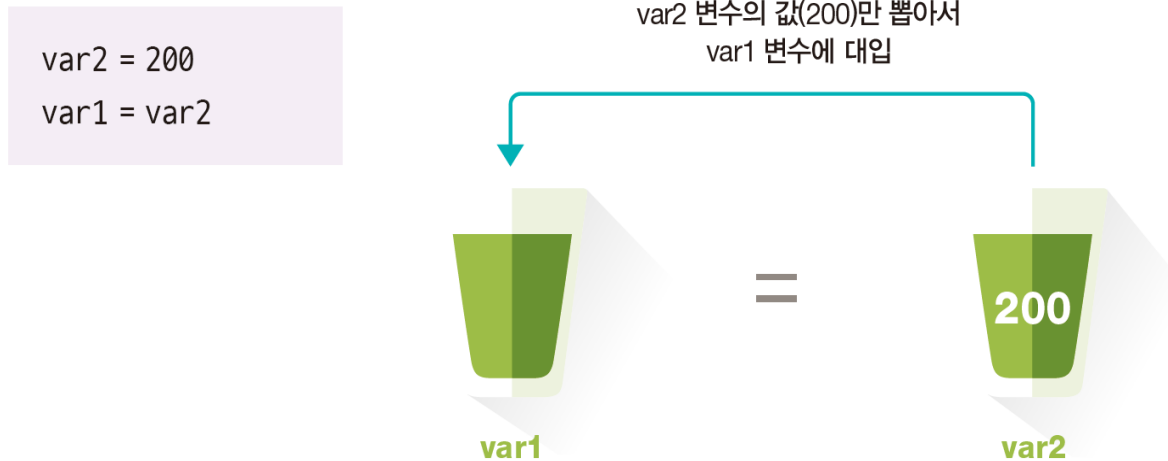


그림 3-9 변수에 변수를 대입하는 방식



## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 변수의 값뿐만 아니라 계산 결과를 넣을 수도 있음

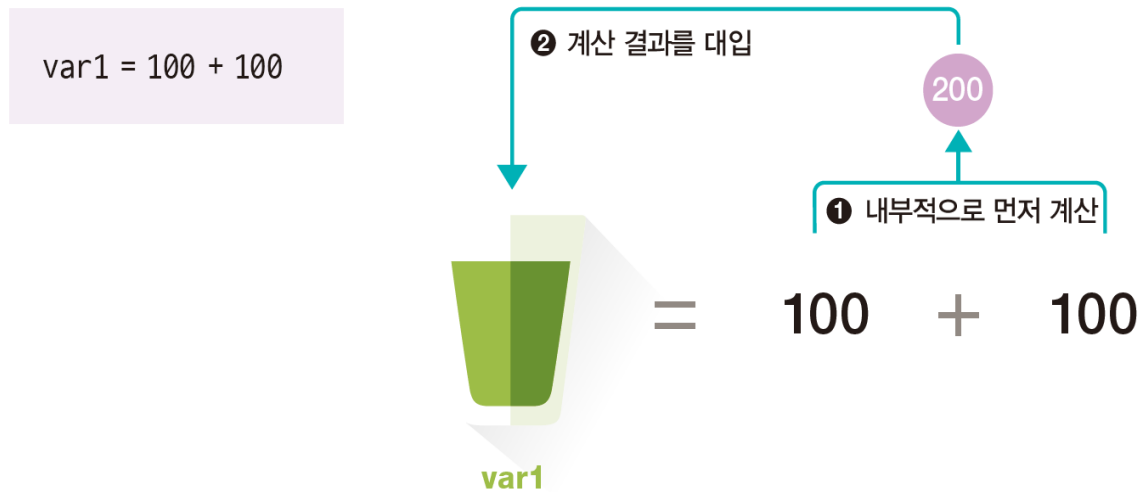


그림 3-10 숫자끼리 연산한 결과를 대입하는 방식

## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 변수의 값뿐만 아니라 계산 결과를 넣을 수도 있음



그림 3-11 변수와 숫자를 연산한 결과를 대입하는 방식

## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 대입 연산자(=)는 맨 뒤부터 처리되므로 왼쪽은 오른쪽과 같이 풀어 쓸 수 있음
- 즉, a, b, c, d 변수에는 모두 100이 대입됨

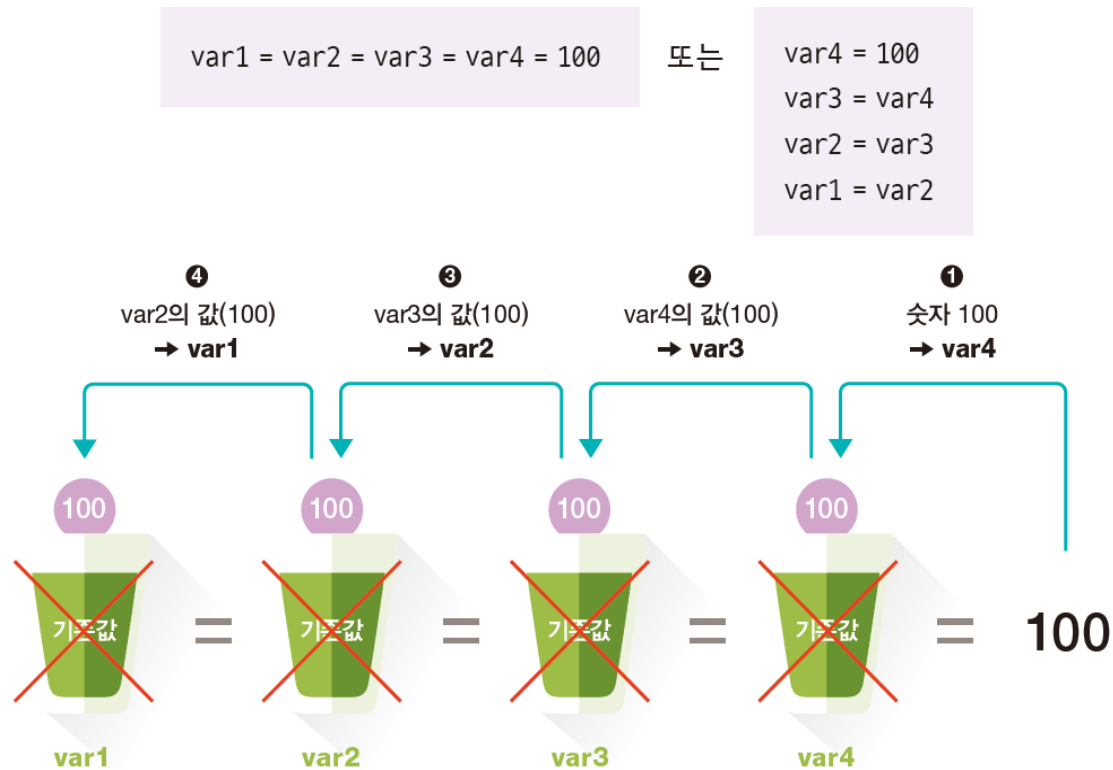


그림 3-12 연속된 값을 대입하는 방식

## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 자신의 값을 연산한 후 이를 다시 자신에게 넣는 방식
- 대입 전 var1에 0 또는 다른 값이 들어 있어야 함을 주의

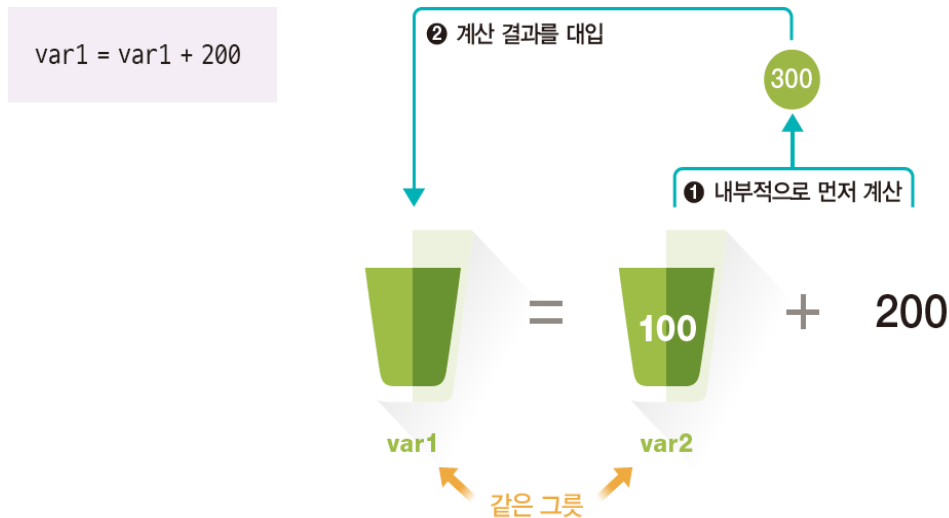


그림 3-13 연산 결과를 자신의 값에 다시 대입하는 방식

## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 아래 noVar 변수에 값이 들어있지 않다면, 'noVar'이 정의되지 않았다는 오류가 발생함
- 이 오류를 방지하려면 코드를 수행하기 전 noVar=0 등의 형태로 값을 대입시켜 놓아야 함

```
noVar = noVar + 200
```

## Section 03 변수의 선언과 사용

### ■ 변수의 사용

- 파이썬에서 변수의 데이터 형식은 값을 넣는 순간마다 변경될 수 있는 유연한 구조

```
myVar = 100          # 정수형 변수를 생성(국그릇 생성)
type(myVar)          # <class 'int'> 출력
myVar = 100.0        # 이 순간에 실수형 변수로 변경(밥그릇으로 변경)
type(myVar)          # <class 'float'> 출력
```

## Section 04 데이터형

### ■ 숫자형

- 숫자형 데이터는 소수점 여부에 따라 크게 정수형과 실수형
- 정수형은 소수점이 없는 수

```
a = 123  
type(a)
```

실행 결과

```
<class 'int'>
```

## Section 04 데이터형

## ■ 숫자형

- 100의 100번 제공한 결과도 잘 대입됨
- 즉, int 크기에 제한이 없음

```
a = 100 ** 100
print(a)
```

## 실행 결과

1000000~00000



## Section 04 데이터형

### ■ 숫자형

- 16진수, 8진수, 2진수도 사용할 수 있다
- 16진수는 0x나 0X로, 8진수는 0o나 0O(숫자+알파벳)로, 2진수는 0b나 0B로 표현

```
a = 0xFF  
b = 0o77  
c = 0b1111  
print(a, b, c)
```

실행 결과

255 63 15

## Section 04 데이터형

### ■ 숫자형

- 실수형은 3.14, -2.7처럼 소수점이 있는 데이터
- 3.14e5처럼 표현할 수도 있음( $3.14e5 = 3.14 * 10^5$ )

```
a = 0xFF  
b = 0o77  
c = 0b1111  
print(a, b, c)
```

실행 결과

255 63 15

## Section 04 데이터형

### ■ 숫자형

- 사칙 연산 수행 가능

```
a = 10; b = 20  
print(a + b, a - b, a * b, a / b)
```

실행 결과

```
30 -10 200 0.5
```

## Section 04 데이터형

### ■ 숫자형

- \*\* - 제곱을 구함
- % - 나머지를 구함
- // - 나눈 후에 소수점을 버림

```
a, b = 9, 2  
print(a ** b, a % b, a // b)
```

실행 결과

81 1 4

## Section 04 데이터형

### ■ 불형

- 참(True)이나 거짓(False)만 저장할 수 있으며, 논리형이라고도 함

```
a = True  
type(a)
```

실행 결과

```
<class 'bool'>
```

## Section 04 데이터형

### ■ 불형

- 비교의 결과를 참이나 거짓으로 저장하는 데 사용

```
a = (100 == 100)
b = (10 > 100)
print(a, b)
```

실행 결과

True False

## Section 04 데이터형

### ■ 문자열

- 문자열은 'abc', "파이썬 만세", "1" 등 문자집합을 의미
- 양쪽을 큰따옴표(" ")나 작은따옴표(' ')로 감싸야 함

```
a = "파이썬 만세"  
a  
print(a)  
type(a)
```

#### 실행 결과

```
'파이썬 만세'  
파이썬 만세  
<class 'str'>
```

## Section 04 데이터형

### ■ 문자열

- 문자열 중간에 작은따옴표(')나 큰따옴표(")를 출력하고 싶다면 다른 따옴표로 묶어 줌

```
"작은따옴표는 ' 모양이다."  
'큰따옴표는 " 모양이다.'
```

실행 결과

```
"작은따옴표는 ' 모양이다."  
'큰따옴표는 " 모양이다.'
```

- 또는 역슬래시(\) 뒤에 큰따옴표(")나 작은따옴표(')를 사용해도 글자로 인식함

```
a = "이건 큰따옴표 \" 모양."  
b = '이건 작은따옴표 \' 모양.'  
print(a, b)
```

실행 결과

```
이건 큰따옴표 " 모양, 이건 작은따옴표 ' 모양.
```



## Section 04 데이터형

### ■ 문자열

- 문자열을 여러 줄로 넣으려면 중간에 \n을 삽입

```
a = '파이썬 \n만세'  
print(a)
```

실행 결과

```
파이썬  
만세
```

## Section 04 데이터형

### ■ 문자열

- 큰따옴표(“)나 작은따옴표(') 3개를 연속해서 묶어도 됨

```
a = """파이썬  
만세"""  
a  
print(a)
```

실행 결과

```
'파이썬\n만세 '  
파이썬  
만세
```

## Section 04 데이터형

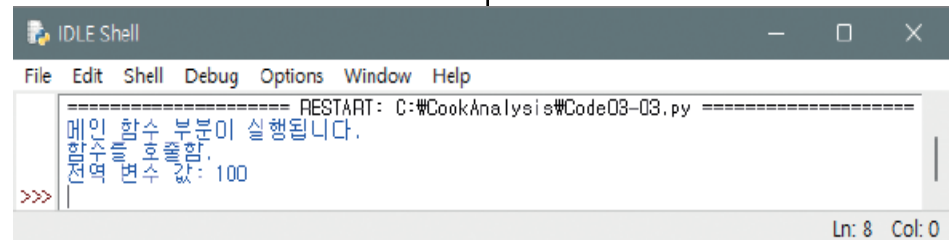
### ■ 문자열

- 파이썬에서는 C/C++, 자바 등과 같은 main() 함수가 존재하지 않는다
- 코드가 길어지면 메인 코드 부분이 혼란스러우므로 main() 함수의 효과를 보여 주는 방법을 이용

Code03-03.py

```
01  ## 함수 선언 부분 ##
02  def myFunc() :
03      print('함수를 호출함.')
04
05  ## 변수 선언 부분 ##
06      gVar = 100

## 메인 코드 부분 ##
if __name__ == '__main__' :
    print('메인 함수 부분이 실행됩니다.')
    myFunc()
    print('전역 변수 값:', gVar)
```



## Section 05 연산자

### ■ 산술 연산자

- 기본적인 계산을 수행할 때 사용하는 연산자

표 3-3 산술 연산자의 종류

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
%	나머지 값	$a = 5 \% 3$	5를 3으로 나눈 후 나머지 값을 a에 대입
**	제곱	$a = 5 ** 3$	5의 3제곱을 a에 대입

## Section 05 연산자

### ■ 산술 연산자

- 기본적인 계산을 수행할 때 사용하는 연산자
- $a//b$ 는  $a$ 를  $b$ 로 나눈 몫이고,  $a\%b$ 는  $a$ 를  $b$ 로 나눈 나머지 값

```
a = 5; b = 3  
print(a + b, a - b, a * b, a / b, a // b, a % b, a ** b)
```

실행 결과

```
8 2 15 1.6666666666666667 1 2 125
```

## Section 05 연산자

## ■ 문자열과 숫자의 상호 변환

- int() 또는 float() 함수를 사용해서 정수나 실수로 변환할 수 있음

[illegible]

## 실행 결과

101 101.123 100000000000000000000000000000000

## Section 05 연산자

### ■ 문자열과 숫자의 상호 변환

- 숫자를 문자열로 변환하려면 `str()` 함수를 사용
- `a`와 `b`가 문자열로 변경되어서 결과가 `100+1`이 아니라 문자열의 연결인 `'1001'`과 `'100.1231'`이 되었음

```
a = 100; b = 100.123  
str(a) + '1'; str(b) + '1'
```

#### 실행 결과

```
'1001'  
'100.1231'
```

## Section 05 연산자

### ■ 대입 연산자

- 연산자 '=' 외에도 다양한 대입 연산자를 사용할 수 있음

표 3-4 대입 연산자의 종류

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

**TIP** • 파이썬에는 C/C++, 자바 등의 언어에 있는 증가 연산자 `++`나 감소 연산자 `--`가 없다.



## Section 05 연산자

### ■ 대입 연산자

- a가 10에서 시작해 프로그램이 진행될수록 값이 누적되는 예시

```
a = 10  
a += 5; print(a)  
a -= 5; print(a)  
a *= 5; print(a)  
a /= 5; print(a)  
a //= 5; print(a)  
a %= 5; print(a)  
a **= 5; print(a)
```

실행 결과

```
15 10 50 10.0 2.0 2.0 32.0
```

## Section 05 연산자

### ■ 관계 연산자

- 비교 연산자라고도 하며 어떤 것이 크거나 작거나 같은지 비교
- 결과는 참(True)이나 거짓(False)이 됨
- 주로 조건문(if 문)이나 반복문(while 문)에서 사용되며, 단독으로는 거의 사용하지 않음

$a < b = \begin{cases} \text{참} & : \text{True} \\ \text{거짓} & : \text{False} \end{cases}$

그림 3-14 관계 연산자의 기본 개념

표 3-5 관계 연산자의 종류

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

## Section 05 연산자

### ■ 관계 연산자

#### ■ 관계 연산자의 예시

```
a, b = 100, 200  
print(a == b , a != b, a > b , a < b , a >= b , a <= b)
```

실행 결과

```
False True False True False True
```

#### ■ 관계 연산자를 착오로 인하여 '=' 하나만 사용한 경우 오류가 나므로 주의

```
print(a = b)
```

# Section 05 연산자

## ■ 논리 연산자

- 'a라는 값이 100과 200 사이에 들어 있어야 한다'는 조건을 표현한 예시

```
(a > 100) and (a < 200)
```

표 3-6 논리 연산자의 종류

연산자	의미	설명	사용 예
and(논리곱)	~이고, 그리고	둘 다 참이어야 참	(a > 100) and (a < 200)
or(논리합)	~이거나, 또는	둘 중 하나만 참이어도 참	(a == 100) or (a == 200)
not(논리부정)	~아니다, 부정	참이면 거짓, 거짓이면 참	not(a < 100)

## Section 05 연산자

### ■ 논리 연산자

- a는 99이므로  $(a > 100)$ 은 거짓,  $(a < 200)$ 은 참이 됨  
즉 '거짓 and 참'이므로 그 결과는 거짓(False)
- 둘 중 하나만 참이어도 참. a는 99이므로  $(a > 100)$ 은 거짓,  
 $(a < 200)$ 은 참이 되어서 '거짓 or 참'이므로 그 결과는 참(True)
- $\text{not}(a == 100)$ 는 a가 99이므로  $(a == 100)$ 은 거짓이 되지만,  
그것의 부정은 참이 되므로 참(True)

```
a = 99
(a > 100) and (a < 200)
(a > 100) or (a < 200)
not(a == 100)
```

실행 결과

False True True

## Section 05 연산자

### ■ 논리 연산자

- 파이썬은 숫자도 참과 거짓으로 구분
- 숫자 0은 거짓(False)이 되고, 그 외의 값은 모두 참(True)으로 취급
- 첫 번째 1234는 참으로 취급하므로 결과가 출력되지만, 두 번째 0은 거짓이므로 결과가 출력되지 않음

```
if(1234) : print("참이면 보여요")  
if(0) : print("거짓이면 안 보여요")
```

## Section 05 연산자

### ■ 연산자 우선순위

- 파이썬에서는 여러 연산자가 동시에 나올 때 어떤 연산자를 먼저 계산할지 이미 결정되어 있음
- 특히 괄호는 최우선이고, 곱셈과 나눗셈이 덧셈과 뺄셈보다 우선하며, 대입 연산자는 늦게 처리된다는 점을 우선적으로 기억하는 것이 좋음

## Section 05 연산자

### ■ 연산자 우선순위

표 3-7 연산자 우선순위

우선순위	연산자	의미
1	()[]{}	괄호, 리스트, 딕셔너리, 세트 등
2	**	지수
3	+ - ~	단항 연산자
4	* / % //	산술 연산자
5	+ -	
6	<< >>	비트 시프트 연산자
7	&	비트 논리곱
8	^	비트 배타적 논리합
9		비트 논리합
10	< > >= <=	관계 연산자
11	== !=	동등 연산자
12	= %= /= //= -= += *= **=	대입 연산자
13	not	논리 연산자
14	and	
15	or	
16	if~else	비교식



## Section 05 연산자

### ■ [프로그램 1] 완성

- 학습한 연산자를 활용한 동전 교환 프로그램 구현

Code03-04.py

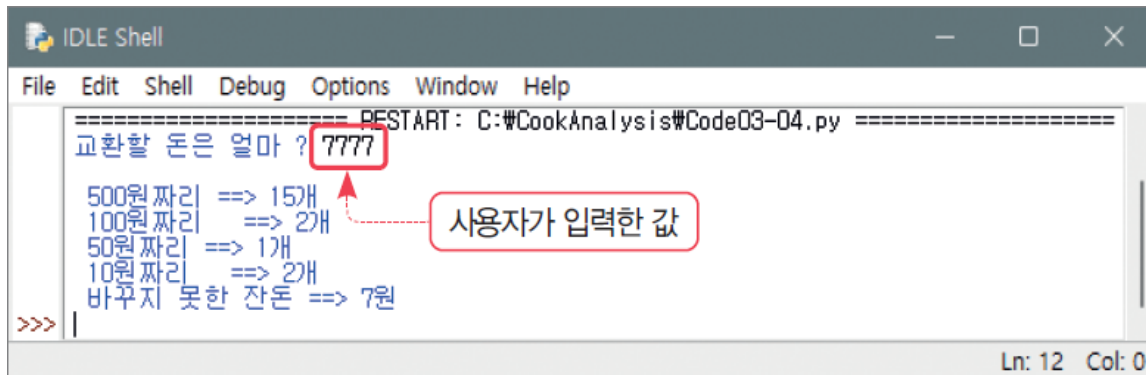
```
01  ## 변수 선언 부분 ##
02  money, c500, c100, c50, c10 = 0, 0, 0, 0, 0
03
04  ## 메인 코드 부분 ##
05  money = int(input("교환할 돈은 얼마?"))
06
07  c500 = money // 500
08  money %= 500
09
10  c100 = money // 100
11  money %= 100
12
13  c50 = money // 50
14  money %= 50
15
16  c10 = money // 10
17  money %= 10
```

## Section 05 연산자

### ■ [프로그램 1] 완성

- 학습한 연산자를 활용한 동전 교환 프로그램 구현

```
18 print("\n 500원짜리 ==> %d개" % c500)
19 print(" 100원짜리 ==> %d개" % c100)
20 print(" 50원짜리 ==> %d개" % c50)
21 print(" 10원짜리 ==> %d개" % c10)
22 print(" 바꾸지 못한 잔돈 ==> %d원 \n" % money)
```



```
===== RESTART: C:\CookAnalysis\Code03-04.py =====
교환할 돈은 얼마 ? 7777
500원짜리 ==> 15개
100원짜리 ==> 2개
50원짜리 ==> 1개
10원짜리 ==> 2개
바꾸지 못한 잔돈 ==> 7원
>>> |
```

사용자가 입력한 값

Ln: 12 Col: 0

## Section 06 조건문

### ■ if 문

- 가장 단순한 형태의 if 문은 참일 때는 뭔가를 실행하고, 거짓일 때는 아무것도 실행하지 않음

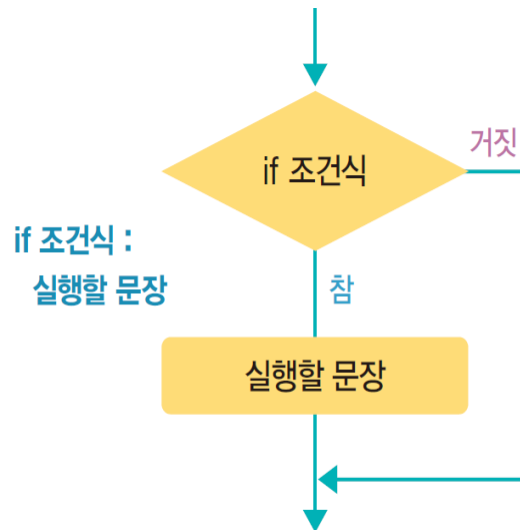


그림 3-15 if 문의 형식과 순서도

## Section 06 조건문

### ■ if 문

- 현재 a에는 99가 들어 있으므로 조건식  $a < 100$ 은 참이 되어 if 문을 실행

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

실행 결과

100보다 작군요.

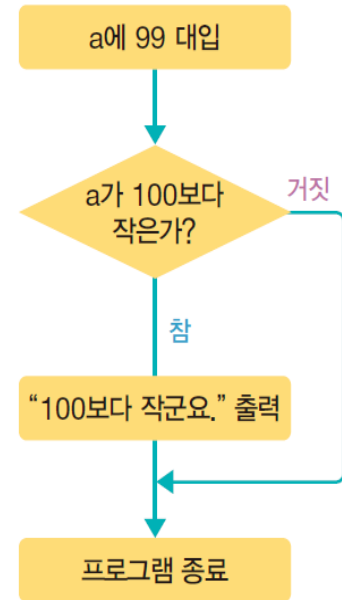


그림 3-16 if 문 실행 과정

## Section 06 조건문

### ■ if 문

- if 문에서 두 문장 이상을 실행하고 싶다면 다음과 같이 실행할 문장을 모두 들여 써야 함

Code03-05.py

```
01 a = 200
02
03 if a < 100 :
04     print("100보다 작군요.")
05     print("거짓이므로 이 문장은 안 보이겠죠?")
06
07 print("프로그램 끝")
```

실행 결과

프로그램 끝

## Section 06 조건문

### ■ if~ else 문

- 참일 때 실행할 문장과 거짓일 때 실행할 문장을 다르게 하려면 if~else 문을 사용

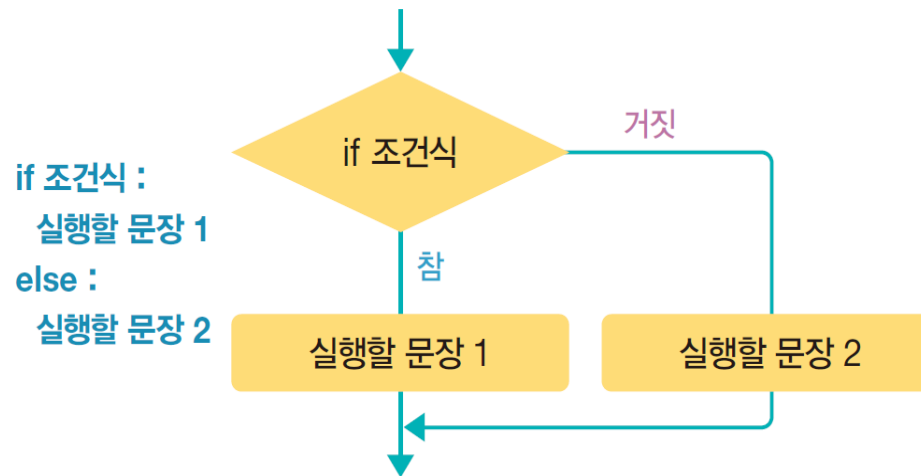


그림 3-17 if~else 문의 형식과 순서도

## Section 06 조건문

### ■ if~ else 문

- 다음과 같이 a에는 200이 들어 있으므로 3행의 조건식은 거짓이 되어 5행의 else 아래에 있는 6행을 실행

Code03-06.py

```
01 a = 200
02
03 if a < 100 :
04     print("100보다 작군요.")
05 else :
06     print("100보다 크군요.")
```

실행 결과

100보다 크군요.

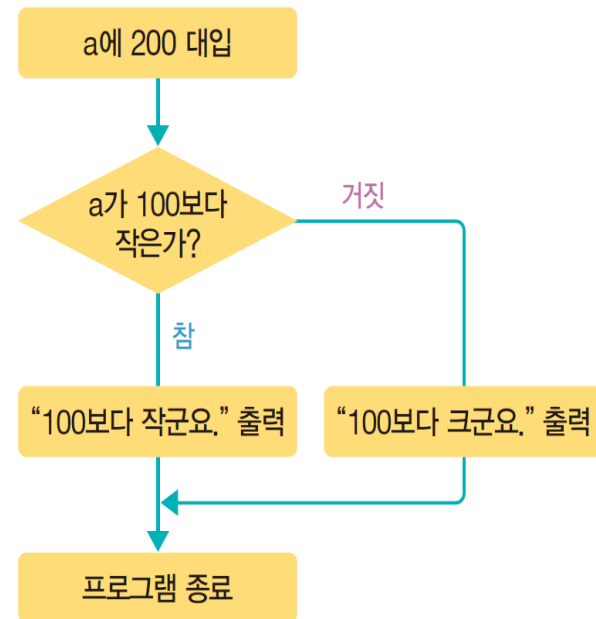


그림 3-18 Code03-06.py 실행 과정

## Section 06 조건문

### ■ if~ else 문

- 입력한 숫자가 짝수인지 홀수인지를 계산하는 코드

Code03-07.py

```
01 a = int(input("정수를 입력하세요 : "))
02
03 if a % 2 == 0 :
04     print("짝수를 입력했군요.")
05 else :
06     print("홀수를 입력했군요.")
```

실행 결과

정수를 입력하세요 : 125  
홀수를 입력했군요.



## Section 06 조건문

### ■ 중첩 if 문

- 예를 들어 강의실에 있는 학생 중에서 20세 이상의 남자가 몇 명인지를 구하는 프로그램을 만든다고 하면, 일단 성별을 구별하고, 남학생 중에서 다시 20대 여부를 구별해야 함
- if 문을 한 번 실행한 후 그 결과에서 if 문을 다시 실행하는 중첩 if 문이 필요함

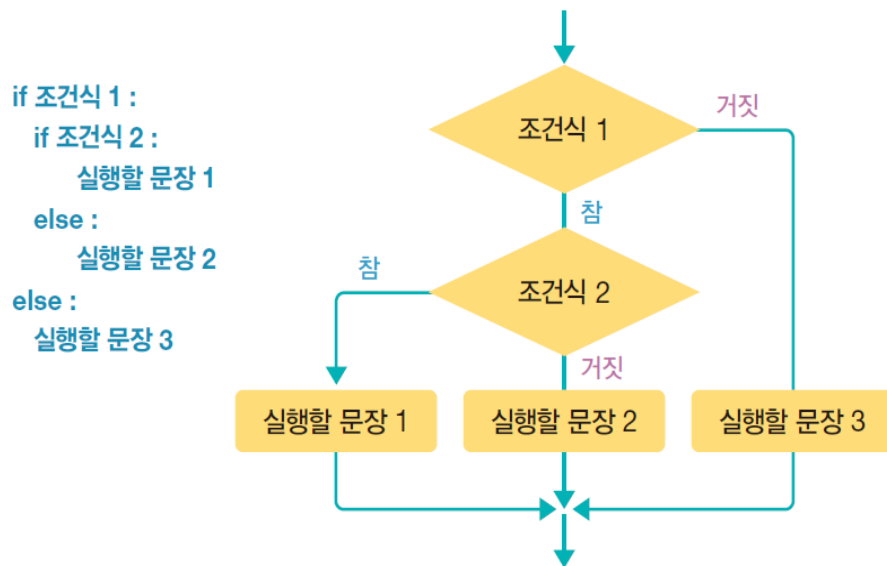


그림 3-19 중첩 if 문의 형식과 순서도

## Section 06 조건문

### ■ 중첩 if 문

Code03-08.py

```
01 a = 75
02
03 if a > 50 :
04     if a < 100 :
05         print("50보다 크고 100보다 작군요.")
06     else :
07         print("와~~ 100보다 크군요.")
08 else :
09     print("에고~ 50보다 작군요.")
```

실행 결과

50보다 크고 100보다 작군요.

## Section 06 조건문

### ■ 중첩 if 문

- 점수를 입력받은 후 90점 이상은 A, 80점 이상은 B, 70점 이상은 C, 60점 이상은 D, 나머지는 F로 처리

Code03-09.py

```
01 score = int(input("점수를 입력하세요 : "))
02
03 if score >= 90 :
04     print("A")
05 else :
06     if score >= 80 :
07         print("B")
08     else :
09         if score >= 70 :
10             print("C")
11         else :
12             if score >= 60 :
13                 print("D")
14             else :
15                 print("F")
16
17 print("학점입니다. ^^")
```

실행 결과

점수를 입력하세요 : 85  
B  
학점입니다. ^^

## Section 06 조건문

### ■ 중첩 if 문

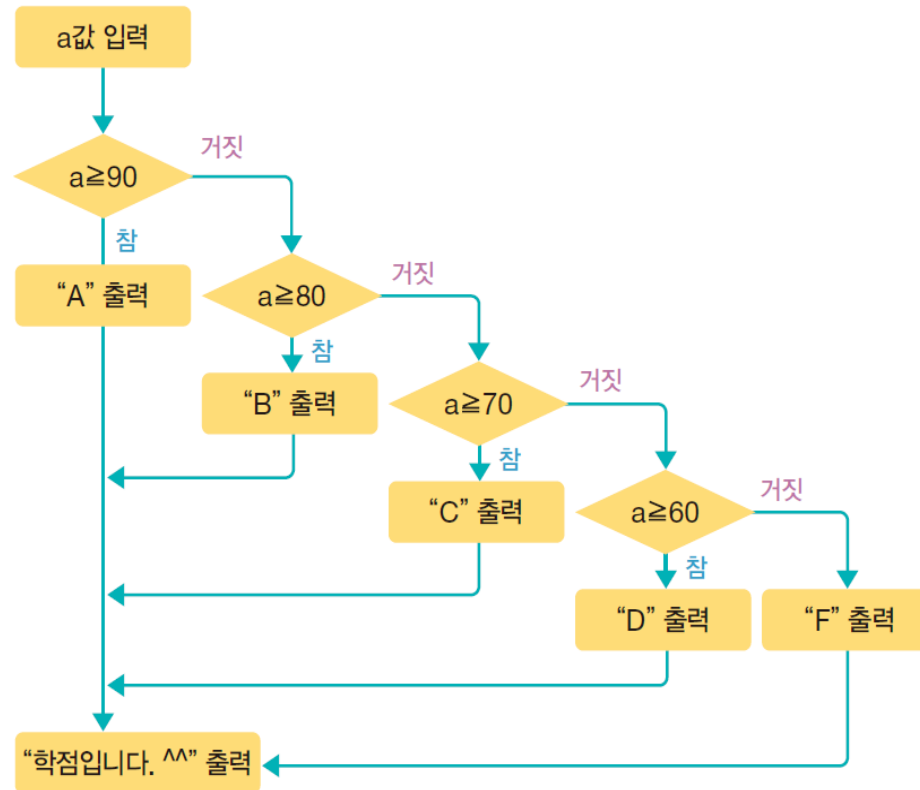


그림 3-20 Code03-09.py 실행 과정

## Section 06 조건문

### ■ if~ elif~ else 문

- Code03-09.py를 if~elif~else 문을 이용하여 Code03-10.py와 같이 수정할 수 있음

Code03-10.py

```
01 score = int(input("점수를 입력하세요 : "))
02
03 if score >= 90 :
04     print("A")
05 elif score >= 80 :
06     print("B")
07 elif score >= 70 :
08     print("C")
09 elif score >= 60 :
10     print("D")
11 else :
12     print("F")
13
14 print("학점입니다. ^^")
```

## Section 06 조건문

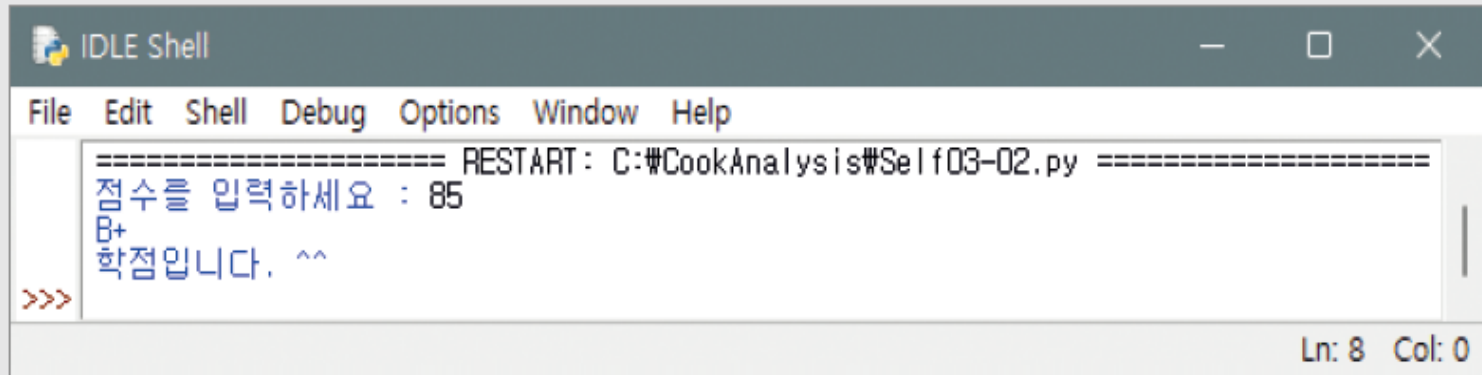
### ■ if~ elif~ else 문

#### SELF STUDY 3-2

Code03-10.py를 다음 조건처럼 좀 더 세분화해 보자.

95점 이상: A+, 90점 이상: A0, 85점 이상: B+, 80점 이상: B0,

75점 이상: C+, 70점 이상: C0, 65점 이상: D+, 60점 이상: D0, 60점 미만: F



```
===== RESTART: C:\#CookAnalysis\Self03-02.py =====
점수를 입력하세요 : 85
B+
학점입니다. ^^
>>>
```

Ln: 8 Col: 0

## Section 06 조건문

### ■ If 문과 리스트

- 리스트(List)는 데이터 여러 개를 한곳에 담아 놓은 것
- 리스트는 대괄호([])로 묶고 그 안에 필요한 것들을 한꺼번에 넣어 만들 수 있음
- 다음 코드는 fruit 변수에 값 4개를 리스트 하나로 묶어 대입한 것

```
fruit = ['사과', '배', '딸기', '포도']  
print(fruit)
```

실행 결과

```
['사과', '배', '딸기', '포도']
```

## Section 06 조건문

### ■ If 문과 리스트

- 리스트의 마지막에 무언가를 더 추가하려면 리스트명.append() 함수를 사용

```
fruit.append('귤')  
print(fruit)
```

실행 결과

```
['사과', '배', '딸기', '포도', '귤']
```



## Section 06 조건문

### ■ If 문과 리스트

- 리스트 안에 무엇이 들어 있는지도 if 문으로 활용할 수 있음
- 'if 항목 in 리스트 :'는 리스트에 해당 항목이 있다면 True를 반환
- 이 코드에서는 딸기가 있으므로 True를 반환해 아래의 print() 함수가 실행됨
- 반대로 'if 항목 not in 리스트 :'는 리스트에 항목이 없어야 True를 반환

```
if '딸기' in fruit :  
    print("딸기가 있네요. ^^")
```

#### 실행 결과

딸기가 있네요. ^^

## Section 06 조건문

### ■ If 문과 리스트

- 0부터 9까지 숫자 중에서 리스트 안에 없는 숫자를 찾아내는 예제
- 2장에서 설명한 random.randrange(시작값, 끝값) 함수를 사용
- 시작값부터 끝값-1까지의 숫자 중에서 임의의 숫자 하나를 반환

Code03-11.py

```
01 import random
02
03 numbers = []
04 for num in range(0, 10) :
05     numbers.append(random.randrange(0, 10))
06
07 print("생성된 리스트", numbers)
08
09 for num in range(0, 10) :
10     if num not in numbers :
11         print("숫자 %d는(은) 리스트에 없네요." %num)
```

#### 실행 결과

생성된 리스트 [5, 8, 8, 7, 8, 1, 9, 0, 0, 4]  
숫자 2는(은) 리스트에 없네요.  
숫자 3는(은) 리스트에 없네요.  
숫자 6는(은) 리스트에 없네요.

## Section 06 조건문

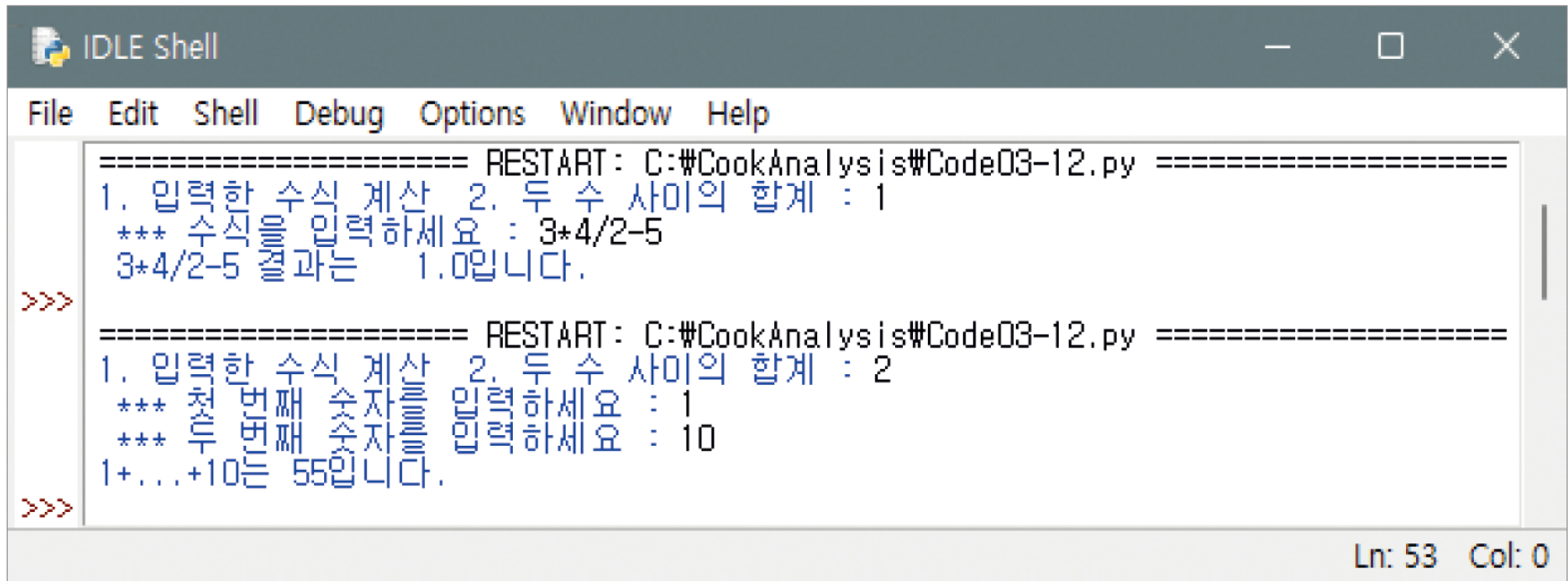
### ■ [프로그램 2] 완성

Code03-12.py

```
01  ## 변수 선언 부분 ##
02  select, answer, numStr, num1, num2 = 0, 0, "", 0, 0
03
04  ## 메인 코드 부분 ##
05  select = int(input("1. 입력한 수식 계산 2. 두 수 사이의 합계 : "))
06
07  if select == 1 :
08      numStr = input(" *** 수식을 입력하세요 : ")
09      answer = eval(numStr)
10      print(" %s 결과는 %5.1f입니다. " % (numStr, answer))
11  elif select == 2 :
12      num1 = int(input(" *** 첫 번째 숫자를 입력하세요 : "))
13      num2 = int(input(" *** 두 번째 숫자를 입력하세요 : "))
14      for i in range(num1, num2 + 1) :
15          answer = answer + i
16      print("%d+...+d는 %d입니다. " % (num1, num2, answer))
17  else :
18      print("1 또는 2만 입력해야 합니다.")
```

## Section 06 조건문

### ■ [프로그램 2] 완성



```
===== RESTART: C:\#CookAnalysis\#Code03-12.py =====
1. 입력한 수식 계산 2. 두 수 사이의 합계 : 1
*** 수식을 입력하세요 : 3*4/2-5
3*4/2-5 결과는 1.0입니다.
>>>

===== RESTART: C:\#CookAnalysis\#Code03-12.py =====
1. 입력한 수식 계산 2. 두 수 사이의 합계 : 2
*** 첫 번째 숫자를 입력하세요 : 1
*** 두 번째 숫자를 입력하세요 : 10
1+...+10는 55입니다.
>>>
```

Ln: 53 Col: 0

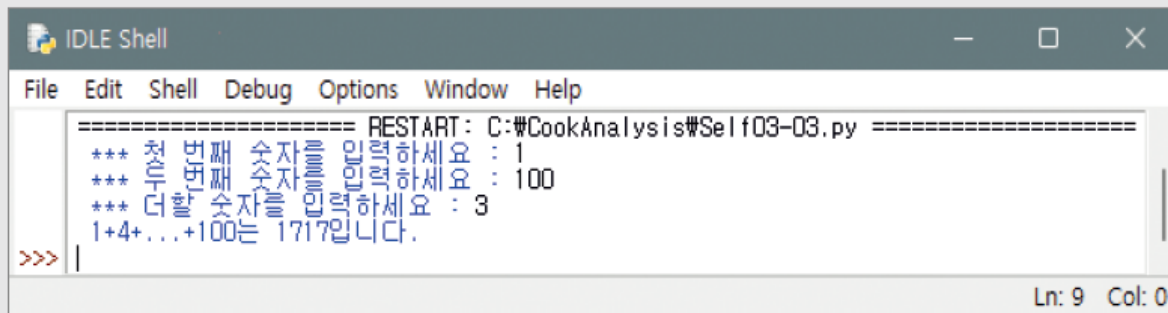
## Section 06 조건문

### ■ [프로그램 2] 완성

#### SELF STUDY 3-3

[프로그램 1]의 두 번째 기능처럼 두 숫자를 입력받고 두 숫자 사이의 합계를 구하는 프로그램을 만들어 보자. 단 1씩 증가하지 않고 증가하는 숫자도 입력받는다. 예를 들어 1, 100, 3을 입력하면  $1+4+\dots+100$ 의 합계를 구한다.

**힌트** range(시작값, 끝값+1, 증가값) 형식으로 사용한다.



```
===== RESTART: C:\#CookAnalysis\Self03-03.py =====
*** 첫 번째 숫자를 입력하세요 : 1
*** 두 번째 숫자를 입력하세요 : 100
*** 더할 숫자를 입력하세요 : 3
1+4+...+100는 1717입니다.
>>> |
```

## Section 07 반복문

### ■ 반복문 개요

- 반복문은 문장을 반복해서 만드는 것
- for 문과 while 문으로 나뉨

#### 실행 결과

```
안녕하세요? for 문을 공부 중입니다. ^^  
안녕하세요? for 문을 공부 중입니다. ^^  
안녕하세요? for 문을 공부 중입니다. ^^
```

- 반복문을 배우기 전 까지는 위와 같은 결과를 출력하기 위해 아래와 같은 코드를 작성할 수 있었을 것임

#### Code03-13.py

```
01 print("안녕하세요? for 문을 공부 중입니다. ^^")  
02 print("안녕하세요? for 문을 공부 중입니다. ^^")  
03 print("안녕하세요? for 문을 공부 중입니다. ^^")
```

## Section 07 반복문

### ■ 반복문 개요

- 그러나 같은 문장을 10,000번 출력해야 한다면 반복문이 필수적임
- 아래는 반복문 중 for 문의 예시

Code03-13(2).py

```
01 for i in range(0, 3, 1) :  
02     print("안녕하세요? for 문을 공부 중입니다. ^^")
```

실행 결과

안녕하세요? for 문을 공부 중입니다. ^^  
안녕하세요? for 문을 공부 중입니다. ^^  
안녕하세요? for 문을 공부 중입니다. ^^

- 1행의 range(0, 3, 1)을 보면, 가운데가 3이므로 3번을 수행할 것임을 미루어 짐작할 수 있음
- 10,000번을 반복해야 한다면 숫자 3 대신에 10000을 입력하면 됨

## Section 07 반복문

### ■ for 문

- for 문의 기본 형식은 다음과 같음

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

- range() 함수는 지정된 범위의 값을 반환
- 앞서 사용한 range(0, 3, 1)은 0에서 시작해 2까지 1씩 증가하는 값을 반환
- 표시된 숫자보다 하나 작은 숫자까지 반환되는 점에 주의



## Section 07 반복문

### ■ for 문

```
for i in range(0, 3, 1) :  
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

- range(0, 3, 1)은 [0, 1, 2]와 같으므로 내부적으로 다음과 같이 변경됨

```
for i in [0, 1, 2] :  
    print("안녕하세요? for 문을 공부 중입니다. ^^")
```

- 그리고 i에 0, 1, 2를 차례로 대입한 후 다음과 같이 3회 반복함
  - 제1회 : i에 0을 대입한 후 print() 함수 수행
  - 제2회 : i에 1을 대입한 후 print() 함수 수행
  - 제3회 : i에 2를 대입한 후 print() 함수 수행

## Section 07 반복문

### ■ for 문

- i 의 값을 직접 사용한 예시

```
for i in range(0, 3, 1) :  
    print("%d : 안녕하세요? for 문을 공부 중입니다. ^^" % i)
```

#### 실행 결과

```
0 : 안녕하세요? for 문을 공부 중입니다. ^^  
1 : 안녕하세요? for 문을 공부 중입니다. ^^  
2 : 안녕하세요? for 문을 공부 중입니다. ^^
```

## Section 07 반복문

### ■ for 문

- for 문의 i를 활용하여 1부터 5까지의 숫자들을 차례로 출력하는 예시

```
for i in range(1, 6, 1) :  
    print("%d " % i, end = " ")
```

실행 결과

1 2 3 4 5

- print() 함수의 마지막에 end=""를 넣어 마지막을 공백으로 처리해 한 칸씩 띄어 출력
- 이것을 삭제하면 print() 함수가 끝나고 자동으로 다음 행으로 넘어감

## Section 07 반복문

### ■ for 문 활용

- 반복적인 덧셈을 수행할 때는 오타의 가능성을 대비하여 반복문을 사용하는 것이 바람직함

1부터 10까지 변할 i 변수 준비

for i 변수가 1을 시작으로 10까지 1씩 증가  
hap값에 i값을 더함

hap값 출력



Code03-14(1).py

```
01 i = 0
02
03 for i in range(1, 11, 1) :
04     hap = hap + i
05
06 print("1에서 10까지의 합계 : %d" % hap)
```

## Section 07 반복문

### ■ for 문 활용

- hap에 어떤 값이 있어야 다시 누적되는데, hap 자체가 존재하지 않아 더할 것이 없어서 오류가 발생함

#### 실행 결과

```
Traceback (most recent call last):  
  File "C:\CookAnalysis\Code03-14(1).py", line 4, in <module>  
    hap = hap + i  
NameError: name 'hap' is not defined
```

## Section 07 반복문

### ■ for 문 활용

- hap에 어떤 값이 있어야 다시 누적되는데, hap 자체가 존재하지 않아 더할 것이 없어서 오류가 발생
- 1행에 hap 을 초기화하는 코드를 추가하면 아래와 같음

Code03-14(2).py

```
01 i, hap = 0, 0
02
03 for i in range(1, 11, 1) :
04     hap = hap + i
05
06 print("1에서 10까지의 합계 : %d" % hap)
```

실행 결과

1에서 10까지의 합계 : 55

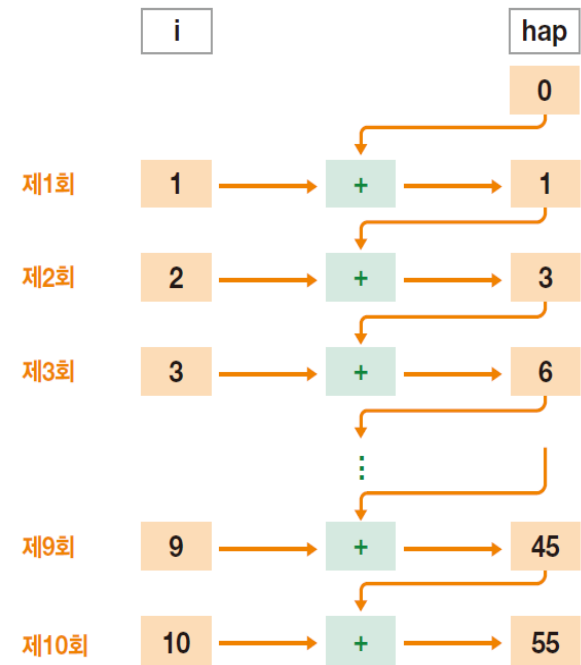


그림 3-21 i와 hap 변수 값의 변화

## Section 07 반복문

### ■ for 문 활용

- 500과 1,000 사이에 있는 홀수의 합계를 구하는 프로그램
- for 문의 범위를 지정하는 range() 함수의 시작값, 끝값+1, 증가값만 적절히 변경하면 다양한 형태의 합계를 구할 수 있음

Code03-15.py

```
01 i, hap = 0, 0
02
03 for i in range(501, 1001, 2) :
04     hap = hap + i
05
06 print("500과 1000 사이에 있는 홀수의 합계 : %d" % hap)
```

실행 결과

500과 1000 사이에 있는 홀수의 합계 : 187500

## Section 07 반복문

### ■ 중첩 for 문

- 중첩 for 문은 for 문 내부에 또 다른 for 문이 들어 있는 형태

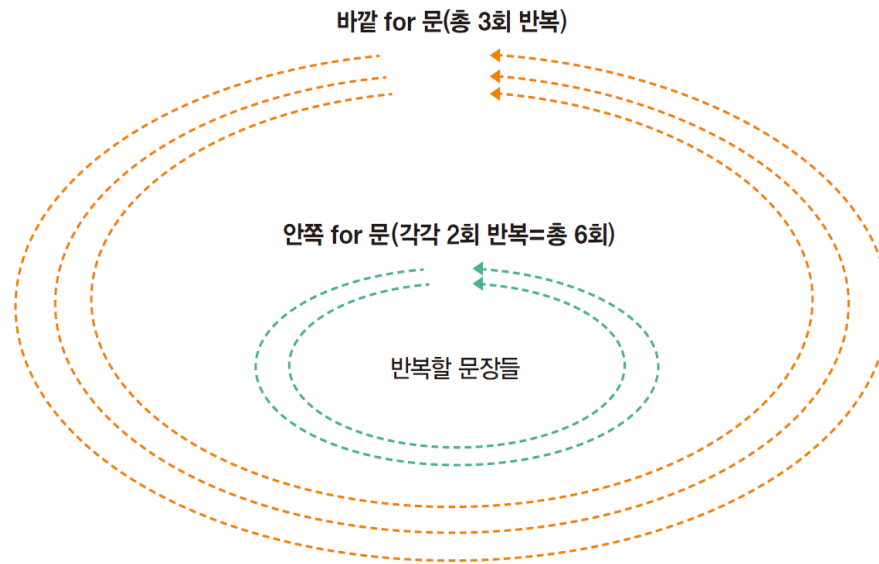


그림 3-22 중첩 for 문의 작동 개념



## Section 07 반복문

### ■ 중첩 for 문

- 중첩 for 문의 기본 형식을 보여 주는 코드

```
for i in range(0, 3, 1):  
    for k in range(0, 2, 1):  
        print("파이썬은 꿀잼입니다. ^^ (i값 : %d, k값 : %d)" % (i, k))
```

#### 실행 결과

```
파이썬은 꿀잼입니다. ^^ (i값 : 0, k값 : 0)  
파이썬은 꿀잼입니다. ^^ (i값 : 0, k값 : 1)  
파이썬은 꿀잼입니다. ^^ (i값 : 1, k값 : 0)  
파이썬은 꿀잼입니다. ^^ (i값 : 1, k값 : 1)  
파이썬은 꿀잼입니다. ^^ (i값 : 2, k값 : 0)  
파이썬은 꿀잼입니다. ^^ (i값 : 2, k값 : 1)
```

## Section 07 반복문

### ■ 중첩 for 문

- 중첩 for 문의 기본 형식을 보여 주는 코드

❶ 외부 for 문 1회 : i에 0을 대입  
내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행  
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

❷ 외부 for 문 2회 : i에 1을 대입  
내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행  
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

❸ 외부 for 문 3회 : i에 2를 대입  
내부 for 문 1회 : k에 0을 대입 후 print() 함수 수행  
내부 for 문 2회 : k에 1을 대입 후 print() 함수 수행

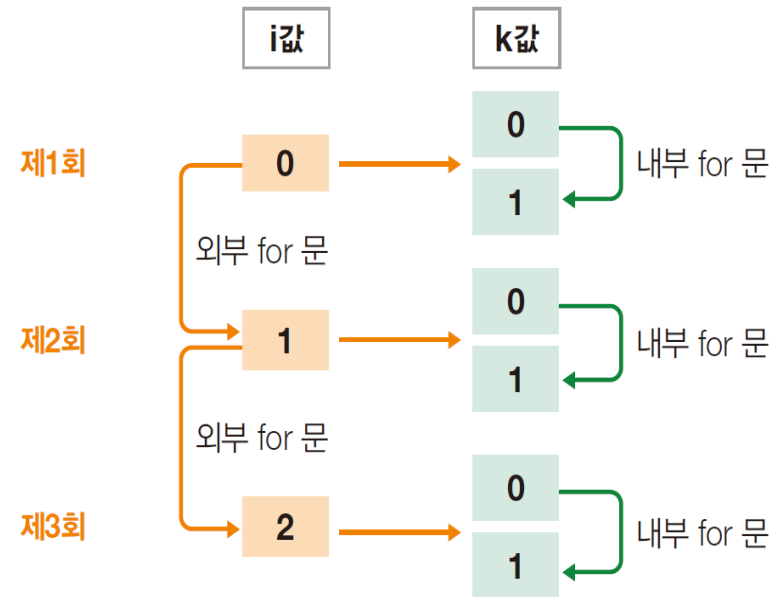


그림 3-23 중첩 for 문에서 i와 k값 변화

## Section 07 반복문

### ■ 중첩 for 문 활용

- 중첩 for 문을 활용해서 2단부터 9단까지 구구단을 출력



그림 3-24 구구단에서 i와 k 변수 추출

## Section 07 반복문

### ■ 중첩 for 문 활용

- 중첩 for 문을 활용한 2단부터 9단까지의 구구단 출력 코드

Code03-16.py

```
01 i, k = 0, 0
02
03 for i in range(2, 10, 1) :
04     for k in range(1, 10, 1) :
05         print("%d X %d = %2d" % (i, k, i * k))
06     print("")
```

실행 결과

```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
... 중략 ...
9 X 8 = 72
9 X 9 = 81
```

## Section 07 반복문

### ■ 중첩 for 문 활용

#### SELF STUDY 3-5

Code03-16.py를 각 단의 제목이 출력되도록 수정해 보자.

##### 실행 결과

```
## 2단 ##  
2 X 1 = 2  
2 X 2 = 4  
2 X 3 = 6  
... 생략 ...
```

## Section 07 반복문

### ■ while 문

- while 문의 기본 형식과 실행 순서

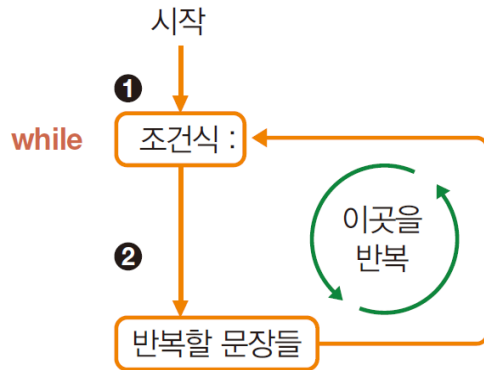


그림 3-25 while 문의 형식과 순서도

- while 문 안의 조건식을 확인해 이 값이 참이면 '반복할 문장들'을 수행
- 그리고 '반복할 문장들'이 끝나는 곳에서 다시 조건식으로 돌아와 같은 동작을 반복
- 조건식이 참인 동안에는 계속 반복한다는 점에 유의

## Section 07 반복문

### ■ while 문

- for 문과 비슷하게 사용할 수 있는 while 문의 형식은 다음과 같음

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

- while 문을 활용하여 '안녕하세요?~' 문장을 3회 출력하는 코드

```
i = 0  
while i < 3 :  
    print("%d : 안녕하세요? while 문을 공부 중입니다. ^^" % i)  
    i = i + 1
```

#### 실행 결과

```
0 : 안녕하세요? while 문을 공부 중입니다. ^^  
1 : 안녕하세요? while 문을 공부 중입니다. ^^  
2 : 안녕하세요? while 문을 공부 중입니다. ^^
```

## Section 07 반복문

### ■ while 문

- Code03-14(2).py에서 for 문으로 작성한 1에서 10까지의 합계 구하기를 while 문으로 구현

Code03-17.py

```
01 i, hap = 0, 0
02
03 i = 1
04 while i < 11 :
05     hap = hap + i
06     i = i + 1
07
08 print("1에서 10까지의 합계 : %d" % hap)
```

실행 결과

1에서 10까지의 합계 : 55



## Section 07 반복문

### ■ while 문 무한반복

- while 문의 조건에 'True'를 넣음
- 무한 루프를 반복하는 간단한 while 문을 실행하면 글자가 무한정 출력되는데, 중지하려면 Ctrl + C 를 누름

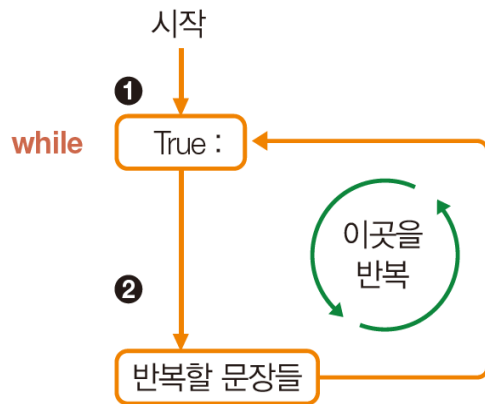


그림 3-26 while 문을 이용한 무한 루프

```
while True :  
    print("ㅋ ", end = " ")
```

#### 실행 결과

ㅋ ㅋ ㅋ ㅋ ㅋ ㅋ ㅋ ㅋ ~~~ 무한 반복

## Section 07 반복문

### ■ break 문

- 앞서 배운 for 문은 range( ) 함수에서 지정한 범위를 벗어나면 종료함
- while 문은 조건식이 거짓이 되면 종료하거나 무한 반복해 Ctrl + C 를 누르면 종료
- break 문은 계속되는 반복을 '논리적으로' 빠져나가는 방법

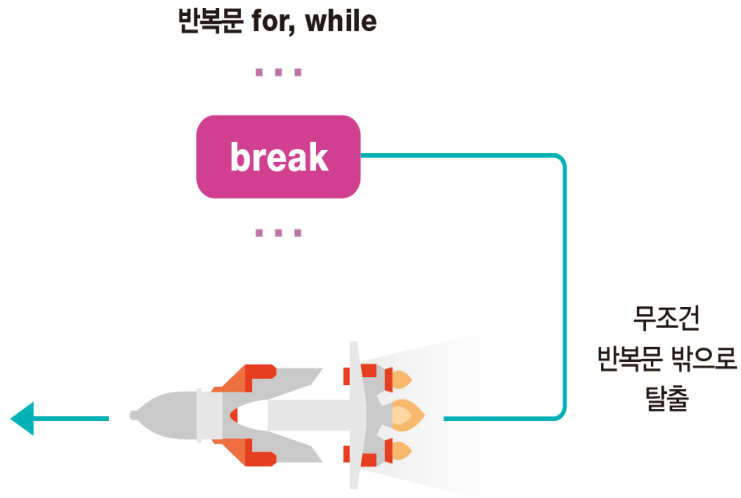


그림 3-27 break 문의 작동

```
for i in range(1, 100) :  
    print("for 문을 %d번 실행했습니다." % i)  
    break
```

#### 실행 결과

for 문을 1번 실행했습니다.

## Section 07 반복문

### ■ break 문

- 사용자가 Ctrl + C 를 누를 때까지 계속 두 수를 더하는 코드를 break 문으로 첫 번째 수에 0이 입력될 때 자동으로 종료되도록 하는 코드

Code03-18.py

```
01 hap = 0
02 a, b = 0, 0
03
04 while True :
05     a = int(input("더할 첫 번째 수를 입력하세요 : "))
06     if a == 0 :
07         break
08     b = int(input("더할 두 번째 수를 입력하세요 : "))
09     hap = a + b
10     print("%d + %d = %d" % (a, b, hap))
11
12 print("0을 입력해 반복문을 탈출했습니다.")
```

#### 실행 결과

더할 첫 번째 수를 입력하세요 : 55  
더할 두 번째 수를 입력하세요 : 22  
55 + 22 = 77  
더할 첫 번째 수를 입력하세요 : 77  
더할 두 번째 수를 입력하세요 : 128  
77 + 128 = 205  
더할 첫 번째 수를 입력하세요 : 0  
0을 입력해 반복문을 탈출했습니다.

## Section 07 반복문

### ■ continue 문

- continue 문을 만나면 블록의 남은 부분을 무조건 건너뛰고 반복문의 처음으로 돌아감
- 즉, 반복문을 처음부터 다시 수행하는 것

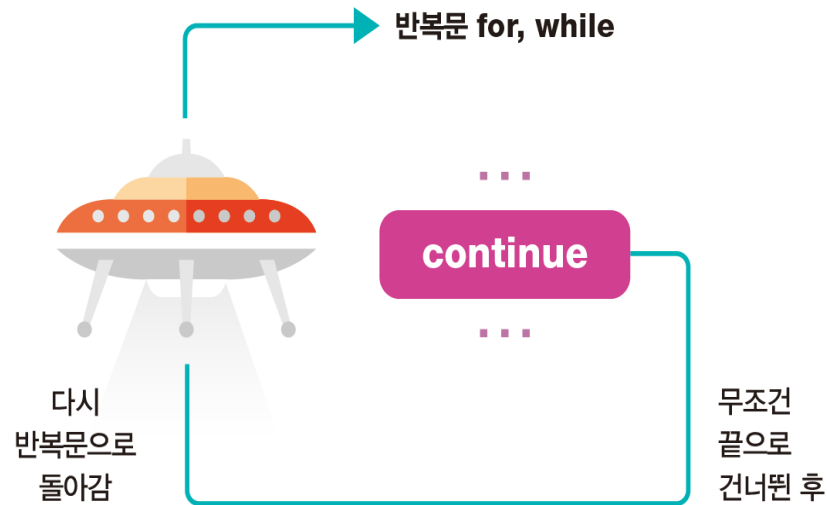


그림 3-28 continue 문의 작동

## Section 07 반복문

### ■ continue 문

- 1부터 100까지의 3의 배수를 제외하고 더하는 프로그램 (1+2+4+5+7+8+10+...)

Code03-19.py

```
01 hap, i = 0, 0
02
03 for i in range(1, 101) :
04     if i % 3 == 0 :
05         continue
06
07     hap += i
08
09 print("1~100의 합계(3의 배수 제외) : %d" % hap)
```

실행 결과

1~100의 합계(3의 배수 제외) : 3367

## Section 07 반복문

### ■ continue 문

- 제1회 :  $i$  값 1을 3으로 나누면 나머지는 1(거짓)  $\rightarrow$   $hap+=1$  수행
- 제2회 :  $i$  값 2를 3으로 나누면 나머지는 2(거짓)  $\rightarrow$   $hap+=2$  수행
- 제3회 :  $i$  값 3을 3으로 나누면 나머지는 0(참)  $\rightarrow$  continue 문 수행
- 다시 3행으로 되돌아가 다음  $i$  값을 준비함
- 제4회 :  $i$  값 4를 3으로 나누면 나머지는 1(거짓)  $\rightarrow$   $hap+=4$  수행
- 제5회 :  $i$  값 5를 3으로 나누면 나머지는 2(거짓)  $\rightarrow$   $hap+=5$  수행
- 제6회 :  $i$  값 6을 3으로 나누면 나머지는 0(참)  $\rightarrow$  continue 문 수행
- 다시 3행으로 되돌아가 다음  $i$  값을 준비함
- 제7회 : ...
- 이렇게 계속 진행하면 계산식  $hap=1+2+4+5+7+\dots$ 이 됨