Python Programming
for Beginners

Computer Engineering
Artificial Intelligence

Creating the Future

03 연산자

2023학년도 2학기
Suk-Hwan Lee

Dong-A University

Division of Computer Engineering &
Artificial Intelligence

산술 연산자의 종류

표 4-1 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
%	나머지값	a = 5 % 3	5를 3으로 나눈 후 나머지값을 a에 대입
**	제곱	a = 5 ** 3	5의 3제곱을 a에 대입

- a//b는 a를 b로 나눈 몫이고, a%b는 a를 b로 나눈 나머지값

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

출력 결과

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

Tip • 세미콜론(;)은 앞뒤를 완전히 분리. 그러므로 a=5; b=3은 다음과 동일하다.
또 콤마(,)로 분리해서 값을 대입할 수도 있어 a, b=5, 3 도 동일

```
a = 5
b = 3
```

■ 산술 연산자의 우선순위

```
a, b, c = 2, 3, 4
print(a + b - c, a + b * c, a * b / c)
```

출력 결과

```
1 14 1.5
```

❶ (a + b) - c
❷ a + (b - c)

- 빨셈에서는 계산되는 순서(연산자 우선순위)가 동일하므로 어떤 것을 먼저 계산하든 동일
- 특별히 괄호가 없을 때는 왼쪽에서 오른쪽 방향으로 계산

산술 연산자의 우선순위

```
❶ (a + b) * c  →  (2 + 3) * 4  →  5 * 4  →  20
❷ a + (b * c)  →  2 + (3 * 4)  →  2 + 12  →  14
```

- 덧셈(또는 뺄셈)과 곱셈(또는 나눗셈)이 같이 있으면 곱셈(또는 나눗셈)이 먼저 계산된 후 덧셈(또는 뺄셈)이 계산
- 괄호가 없어도 ❷처럼 계산
- 산술 연산자는 괄호가 가장 우선, 곱셈(또는 나눗셈)이 그 다음, 덧셈(또는 뺄셈)이 마지막
- 덧셈(또는 뺄셈)끼리 있거나 곱셈(또는 나눗셈)끼리 있으면 왼쪽에서 오른쪽으로

Tip • 덧셈, 뺄셈, 곱셈, 나눗셈이 함께 나오면 연산자 우선순위 때문에 종종 혼란스럽게 느껴진다. 이때는 괄호를 사용하면 된다. 괄호를 사용하면 무조건 괄호가 우선 계산, 두 번째 것이 더 나은 코딩

```
❶ a = b + c * d;
❷ a = b + (c * d);
```

■ 산술 연산을 하는 문자열과 숫자의 상호 변환

- 문자열이 숫자로 구성되어 있을 때, int() 또는 float() 함수 사용해서 정수나 실수로 변환
- 문자열을 int() 함수가 정수로, float() 함수가 실수로 변경

```
s1, s2, s3 = "100", "100.123", "99999999999999999999999999"
print(int(s1) + 1, float(s2) + 1, int(s3) + 1)
```

출력 결과

```
101 101.123 100000000000000000000000000
```

- 숫자를 문자열로 변환하려면 str() 함수 사용.
- a와 b가 문자열로 변경되어 100+1이 아닌 문자열의 연결인 '1001'과 '100.1231' 됨

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

출력 결과

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

Tip • print() 함수는 출력 결과에 작은따옴표가 없어 문자열인지 구분하기가 어려워 사용하지 않음

산술 연산자와 대입 연산자

- 대입 연산자 = 외에도 +=, -=, *=, /=, %=, //=, **= 사용
- 예) 첫 번째 대입 연산자 a+=3은 a에 3을 더해서 다시 a에 넣으라는 의미로 a=a+3과 같음

Tip • 파이썬에는 C/C++, 자바 등의 언어에 있는 증가 연산자 ++나 감소 연산자 --가 없음

표 4-2 대입 연산자의 종류

연산자	사용 예	설명
+=	a += 3	a = a + 3과 동일
-=	a -= 3	a = a - 3과 동일
*=	a *= 3	a = a * 3과 동일
/=	a /= 3	a = a / 3과 동일
//=	a //= 3	a = a // 3과 동일
%=	a %= 3	a = a % 3과 동일
**=	a **= 3	a = a ** 3과 동일

- a가 10에서 시작해 프로그램이 진행될수록 값이 누적

```
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)
```

출력 결과

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

```
PS C:\Users\sky\Documents\Python Scripts\Python-Lecture>
Lecture/CookPython(2019.10.15)/Code04-01.py"
교환할 돈은 얼마 ? 1881

500원짜리 ==> 3개
100원짜리   ==> 3개
50원짜리 ==> 1개
10원짜리   ==> 3개
바꾸지 못한 잔돈 ==> 1원
```

[동전교환 프로그램]의 완성

- 학습한 연산자를 활용해서 동전 교환 프로그램 구현

Code04-01.py

```
1  ## 변수 선언 부분 ##
2  money, c500, c100, c50, c10 = 0, 0, 0, 0, 0
3
4  ## 메인 코드 부분 ##
5  money = int(input("교환할 돈은 얼마?"))
6
7  c500 = money // 500
8  money %= 500
9
10 c100 = money // 100
11 money %= 100
12
13 c50 = money // 50
14 money %= 50
```

2행 : 동전으로 교환할 돈(money)과 500원, 100원, 50원, 10원짜리 동전의 개수를 저장할 변수 초기화

7행 : 500원짜리 동전의 개수를 구함

8행 : 다시 money를 500으로 나눈 후 나머지 값 저장, 8행의 money%=500은 money=money%500과 동일.

10~11행 : 100원짜리 동전을, 13~14행에서 50원짜리 동전을, 16~17행에서 10원짜리 동전을 구함

```
15
16  c10 = money // 10
17  money %= 10
18
19  print("\n 500원짜리 ==> %d개" % c500)
20  print(" 100원짜리  ==> %d개" % c100)
21  print(" 50원짜리 ==> %d개" % c50)
22  print(" 10원짜리  ==> %d개" % c10)
23  print(" 바꾸지 못한 잔돈 ==> %d원 \n" % money)
```

마지막 money에 저장된 값은 10 미만으로, 바꿀 수 없는 나머지 돈

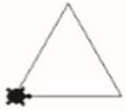
LAB3-1 거북이 그래픽으로 숫자를 입력받아 다각형을 그리자

다음과 같이 사용자가 몇각형인지를 입력하면 해당되는 다각형을 화면에 그리는 프로그램을 작성해 보자.

원하는 결과

```
몇각형을 그리시겠어요?(3-6): 6
```

```
몇각형을 그리시겠어요?(3-6): 3
```

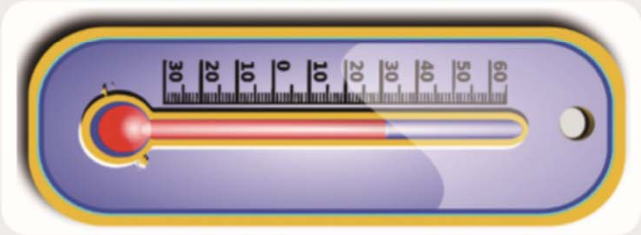


```
import turtle
t = turtle.Turtle()
t.shape("turtle")
n = int(input("몇각형을 그리시겠어요?(3-6): "))

for i in range(n) :     # n회만큼 아래의 두 문장을 반복수행한다
    t.forward(100)
    t.left(360 // n)      # 여기서 나눗셈 연산자인 //을 사용한다

turtle.done()
```

LAB3-2 화씨온도를 섭씨온도로 변환하기

우리나라에서는 온도를 나타낼 때 섭씨온도를 사용하지만 미국에서는 **화씨온도**fahrenheit를 사용한다. 화씨온도를 받아서 섭씨온도로 바꾸는 프로그램을 작성해보자. 화씨온도를 F라고 하면 섭씨온도 C로 바꾸는 식은 다음과 같다.

$$C = (F - 32) \times \frac{5}{9}$$



원하는 결과

```
화씨온도: 100
섭씨온도: 37.77777777777778
```

```
fahrenheit = int(input("화씨온도: "))
celsius = (fahrenheit - 32) * 5 / 9
print("섭씨온도:", celsius)
```

혹은 아래와 같이 실수로 처리해도 된다. 위의 코드가 제대로 동작한 이유는 나눗셈이 사용되면 무조건 실수로 바뀌게 되고, 정수와 실수의 곱도 실수로 간주되게 때문에 최종적으로 실수 값이 나오게 된다.

```
fahrenheit = float(input("화씨온도: "))
celsius = (fahrenheit - 32.0) * 5.0 / 9.0
print("섭씨온도:", celsius)
```

LAB3-3 몸무게와 키를 입력받아 BMI 계산하기

BMIBody Mass Index는 체중(kg)을 신장(m)의 제곱으로 나눈 값으로 체지방 축적을 잘 반영하기 때문에 비만도 판정에 많이 사용한다. 앞 장에서도 BMI 계산을 해보았지만, 변수에 값을 바로 지정해서 결과를 얻었다. 이번에는 사용자의 입력을 받아 처리하도록 개선해 보자. BMI 지수는 킬로그램 단위로 측정한 체중을 w, 미터 단위로 측정한 키가 h라고 할 때 다음과 같이 구할 수 있다.

$$BMI = \frac{w}{h^2}$$

원하는 결과

```
몸무게를 kg 단위로 입력하시오: 95
키를 미터 단위로 입력하시오: 1.82
당신의 BMI= 28.680111097693512
```

```
weight = float(input("몸무게를 kg 단위로 입력하시오: "))
height = float(input("키를 미터 단위로 입력하시오: "))
bmi = (weight / (height ** 2))
print("당신의 BMI=", bmi)
```

도전 문제

도전문제 3.2

BMI는 비만도를 추정하는 데에 가장 흔히 사용되는 지수이지만, 정확도에 한계가 있다는 비판이 많다. BMI 처럼 간단하면서 비만도 계산의 정확도가 높은 것으로 알려진 새로운 지수가 있다. **상대적 지방 질량**relative fat mass 지수로 간단히 줄여서 **RFM**이라고 한다. 남자와 여자에 적용되는 수식이 다르다.

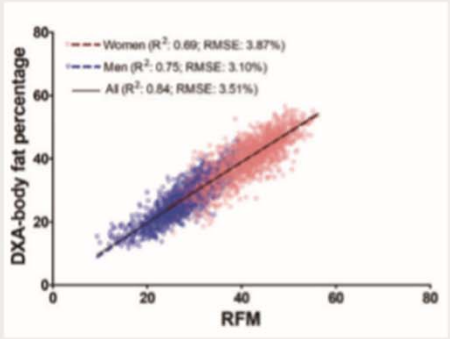
여성을 위한 공식: 76 - (20 × (신장/허리둘레))
남성을 위한 공식: 64 - (20 × (신장/허리둘레))

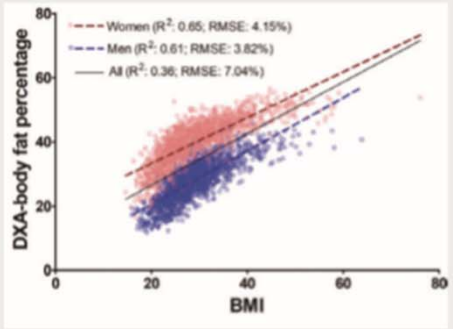
여러 논문을 통해 오른쪽 그림처럼 RFM 지수가 BMI 지수보다 실제 체지방 비율과 더 잘 일치한다고 보고되었다.
남녀 모두를 위한 공식은 성별에 여성은 1, 남성은 0을 적용하여 다음과 같이 구할 수 있다.

RFM = 64 - (20 × (신장 / 허리둘레)) + 12 × 성별

RFM을 계산하는 프로그램을 다음과 같은 방식으로 동작하게 만들어 보라.

```
여성이면 1, 남성이면 0을 입력하세요: 0
당신의 키는 얼마입니까? 181
당신의 허리 둘레는 얼마입니까? 94
당신의 RFM은  25.48936170212766
```





출처: Woolcott OO, Bergman RN (2018) Relative fat mass (RFM) as a new estimator of whole-body fat percentage horizontal line – A cross-sectional study in American adult individuals. Scientific Reports 8(1): 1-11, 2018.

관계연산자의 개념

- 어떤 것이 크거나 작거나 같은지 비교하는 것(참은 True로, 거짓은 False로 표시)
- 주로 조건문(if)이나 반복문(while)에서 사용, 단독으로는 거의 사용하지 않음

a 〈 b = { 참 : True / 거짓 : False

그림 4-1 관계 연산자의 기본 개념

표 4-3 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

- a==b를 보면 100이 200과 같다는 의미이므로 결과는 거짓(False)

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

출력 결과

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

- a와 b를 비교하는 관계 연산자 ==를 사용하려다 착오로 =을 하나만 쓴 코드
  - 빨간색 오류로 나타남. a=b는 b 값을 a에 대입하라는 의미이지 관계 연산자가 아님

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

논리 연산자의 종류와 사용

- and(그리고), or(또는), not(부정) 세 가지 종류
- 예) a라는 값이 100과 200 사이에 들어 있어야 한다는 조건 표현

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

표 4-4 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)

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

출력 결과

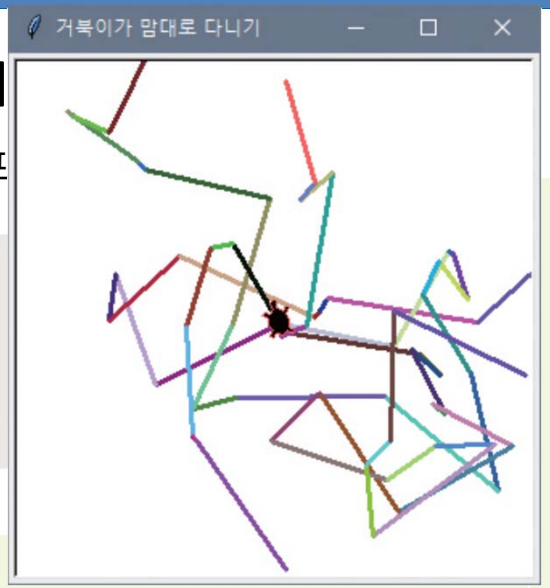
```
False True True
```

- 각 행의 끝에서 Enter 를 2번 눌러야 한다. 첫 번째 1234는 참으로 취급하므로 결과 출력
- 두 번째 0은 거짓이므로 결과가 출력되지 않음. 결론적으로 0은 False, 그 외의 숫자는 모 두 True

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

Lab

- [자유 이동하는 프로그램]
  - 마음대로 이동하는 거북이 프




```
if 조건식 :
  참일 때 수행
else :
  거짓일 때 수행
```

Code04-02.py

```
1  import turtle
2  import random
3
4  ## 전역 변수 선언 부분 ##
5  swidth, sheight, pSize, exitCount = 300, 300, 3, 0
6  r, g, b, angle, dist, curX, curY = [0] * 7
7
```

5~6행 : 변수 준비
- swidth, sheight는 윈도창의 폭과 높이, pSize는 펜의 두께 준비
- exitCount는 윈도창 밖으로 빠져나간 횟수를 위해서 준비.
- r, g, b는 색상, angle과 dist는 임의로 이동할 거리와 각도,
- curX와 curY는 현재 거북이의 위치를 지정하는 변수

```
8  ## 메인 코드 부분 ##
9  turtle.title('거북이가 맘대로 다니기')
10 turtle.shape('turtle')
11 turtle.pensize(pSize)
12 turtle.setup(width = swidth + 30, height = sheight + 30)
13 turtle.screensize(swidth, sheight)
14
15 while True :
16     r = random.random()
17     g = random.random()
18     b = random.random()
19     turtle.pencolor((r, g, b))
20
21     angle =  random.randrange(0, 360)
22     dist = random.randrange(1, 100)
23     turtle.left(angle)
24     turtle.forward(dist)
25     curX = turtle.xcor()
```

창의 제목
거북이 모양
펜 두께 (3)
윈도창 크기 (330, 330)
안쪽 화면 크기 지정 (300, 300)

15~37행 : while True : 문장으로 무한 반복

16~19행 : 임의의 색상 설정

21행과 22행 : 각도(angle)는 0~360 범위에서, 거리 (dist)는 1~100 범위에서 임의 추출

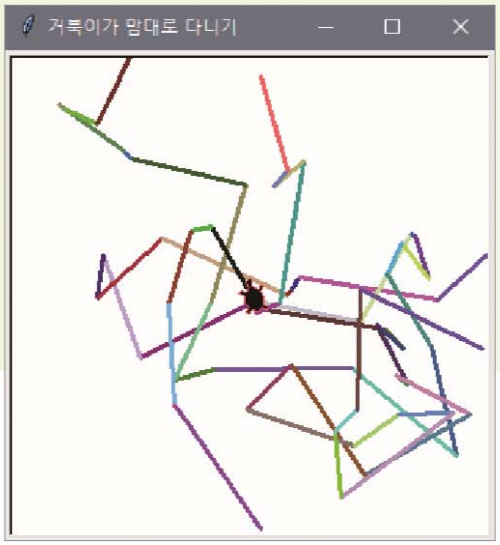
23~24행 : 거북이의 각도 설정 후 거리만큼 이동
25~26행 : 거북이의 현재 위치 구함

Lab

```
26      curY = turtle.ycor()
27
28      if (-swidth / 2 <= curX and curX <= swidth / 2) and (-sheight / 2 <= curY and curY <=
        sheight / 2) :
29          pass
30      else :
31          turtle.penup()
32          turtle.goto(0, 0)
33          turtle.pendown()
34
35          exitCount += 1
36          if exitCount >= 5 :
37              break
38
39  turtle.done()
```

28행은 거북이의 현재 위치가 화면 안인지 체크, 터틀 그래픽의 좌표는 중앙이 (0, 0)

29행 : pass 실행해서 if 문을 그냥 종료하고 다시 while 문 수행 이 범위를 벗어난다면 30~37행을 수행
31행 : 펜 사용하지 않음
32행 : 화면의 중앙으로 이동
33행 : 다시 펜 사용



35행 : 거북이가 바깥으로 나간 횟수를 하나 증가
36행 : 5회 이상 밖으로 나갔다면 break 문으로 while 문을 빠져나간 후 프로그램 종료

Lab

```python
import turtle
import random

## 전역 변수 선언 부분 ##
swidth, sheight, pSize, exitCount = 300, 300, 3, 0
r, g, b, angle, dist, curX, curY = [0] * 7

## turtle 환경 설정 ##
turtle.title('거북이가 맘대로 다니기')
turtle.setup(width=swidth+30, height=sheight+30)
turtle.screensize(swidth, sheight)

## turtle 객체 생성
myT = turtle.Turtle(shape="turtle")
myT.pensize(pSize)

while True :
    r = random.random()
    g = random.random()
    b = random.random()
    myT.pencolor((r, g, b))

    angle =  random.randrange(0,360)
    dist = random.randrange(1,100)
    myT.left(angle)
    myT.forward(dist)
    curX = myT.xcor()
    curY = myT.ycor()

    if (-swidth / 2 <= curX and curX <= swidth / 2) and (-sheight / 2<= curY and curY <= sheight / 2) :
        pass
    else :
        myT.penup()
        myT.goto( 0, 0 )
        myT.pendown()

        exitCount += 1
        if exitCount >= 5 :
            break

turtle.done()
```

비트 연산자의 개념

- 정수를 2진수로 변환한 후 각 자리의 비트끼리 연산 수행
- 비트 연산자의 종류 : &, |, ^, ~, 〈〈, 〉〉

표 4-5 비트 연산자의 종류

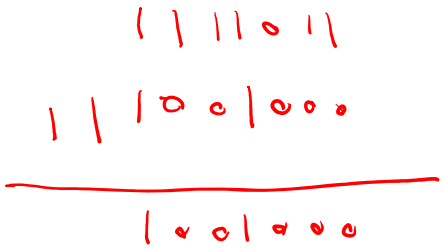
연산자	의미	설명
&	비트 논리곱(and)	둘 다 1이면 1
|	비트 논리합(or)	둘 중 하나만 1이면 1
^	비트 논리적 배타합(xor)	둘이 같으면 0, 다르면 1
~	비트 부정	1은 0으로, 0은 1로 변경
〈〈	비트 이동(왼쪽)	비트를 왼쪽으로 시프트(Shift)
〉〉	비트 이동(오른쪽)	비트를 오른쪽으로 시프트(Shift)

- 123&456은 123의 2진수인 1111011_2와 456의 2진수인 111001000_2의 비트 논리곱(&) 결과인 1001000_2가 되므로 10진수로 72가 나옴
- 두 수의 자릿수가 다를 때는 빈 자리에 0을 채운 후 비트 논리곱 연산
- 0과 비트 논리곱을 수행하면 어떤 숫자든 무조건 0가 된다

```
10 & 7
123 & 456
0xFFFF & 0x0000
```

출력 결과

```
2 72 0
```



비트 논리곱과 비트 논리합 연산자

- and는 그 결과가 참(True) 또는 거짓(False), &는 비트 논리곱을 수행한 결과가 나옴
- 비트 연산은 0과 1밖에 없으므로 0은 False, 1은 True
- 10&7의 결과는 2. [그림 4-2]와 같이 10진수를 2진수로 변환한 후 각 비트마다 and 연산을 수행하기 때문. 그 결과 2진수로는 00102가 되고, 10진수로는 2가 된다

A	B	A&B
0	0	0
0	1	0
1	0	0
1	1	1

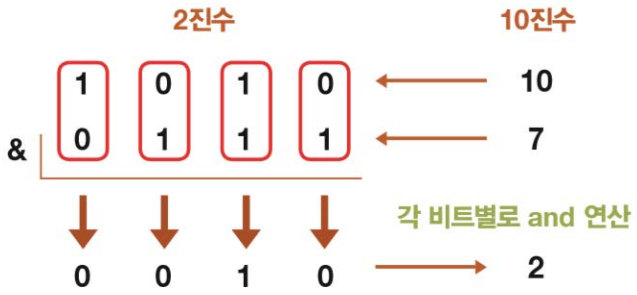


그림 4-2 비트 논리곱의 예

- 10|7과 123|456은 주어진 수의 비트 논리합 연산 수행한 것
- 0xFFFF|0x0000을 보면 0xFFFF와 0000의 비트 논리합은 0xFFFF가 됨. 그러므로 16진수 FFFF16는 10진수 65535가 됨. 여기서 16진수로 출력 원하면 hex(0xFFFF|0x0000) 함수 사용

```
10 ¦ 7
123 ¦ 456
0xFFFF ¦ 0x0000
```

출력 결과

```
15 507 65535
```

- 비트 배타적 논리합 : 두 값이 다르면 1, 같으면 0

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

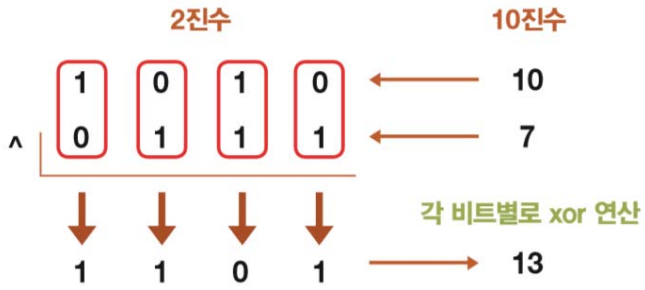


그림 4-4 비트 배타적 논리합의 예

```
10 ^ 7
123 ^ 456
0xFFFF ^ 0x0000
```

출력 결과

```
13 435 65535
```

- 비트 연산 활용 예제

Code04-03.py

```
1  a = ord('A')
2  mask = 0x0F
3
4  print("%x & %x = %x" % (a, mask, a & mask))
5  print("%X & %X = %X" % (a, mask, a | mask))
6
7  mask = ord('a') - ord('A')
8
9  b = a ^ mask
10 print("%c ^ %d = %c" % (a, mask, b))
11 a = b ^ mask
12 print("%c ^ %d = %c" % (b, mask, a))
```

출력 결과

```
41 & f = 1
41 & F = 4F
A ^ 32 = a
a ^ 32 = A
```

- ord() 함수 : 특정한 한 문자를 아스키 코드 값으로 변환해 주는 함수
- chr() 함수 : 아스키 코드 값을 문자로 변환해 주는 함수 (10진수, 16진수 사용 가능)

- Code04-03.py는 마스크(Mask) 방식에 대한 예제(마스크는 무엇을 걸러 주는 역할)
- 우선 마스크값을 2행에서 16진수 0x0F로 선언. 이는 2진수로 0000 1111 의미
- 이것과 비트 논리곱(&) 연산을 하면 [그림 4-5]와 같음

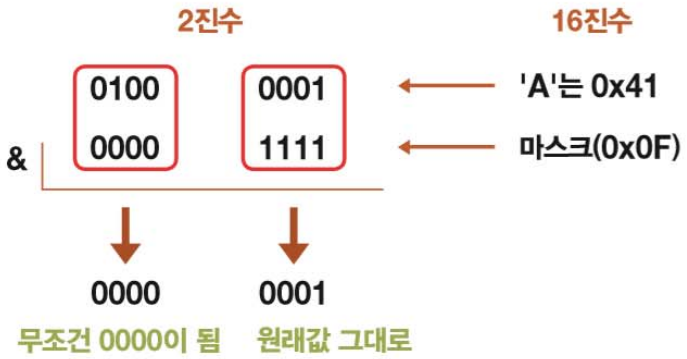


그림 4-5 마스크 0x0F를 사용한 비트 논리곱 결과

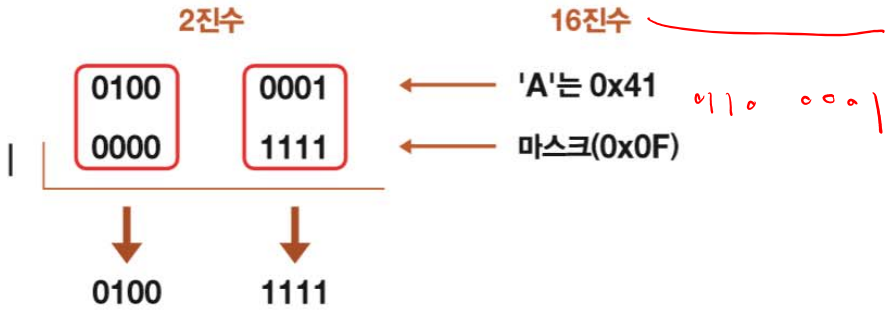


그림 4-6 마스크 0x0F를 사용한 비트 논리합 결과

- 반면 5행처럼 0x0F로 비트 논리합 연산을 하면 [그림 4-6]과 같이 앞 4비트는 원래값 남고, 뒤 4비트는 무조건 1111이 됨
- 7행에서는 마스크값을 소문자(a)와 대문자(A)의 차이로 선언. a는 0x61이고, A는 0x41이므 로 두 값의 차이는 0x20이 됨. 이는 10진수로 32이고, 2진수로는 0010 0000. 9행에서 A 와 32(0010 0000_2)의 비트 배타적 논리합 수행하면 a로 변경, 11행에서 다시 a와 32(0010 0000_2)의 비트 배타적 논리합 수행하면 A로 원상 복귀

Section 04 비트 연산자

- 비트 부정 연산자(또는 보수 연산자) : 두 수를 연산하는 것이 아니라, 하나만 가지고 각 비트를 반 대로 만드는 연산
- 반전된 값을 1의 보수라 하고, 그 값에 1을 더한 값을 2의 보수라고 한다.
- 해당 값의 음수(-)값을 찾고자 할 때 사용
- 정수값에 비트 부정을 수행한 후 1을 더하면 해당 값의 음수값을 얻는 코드

```
a = 12345
~a + 1
```

출력 결과

```
-12345
```

시프트 연산자

- 왼쪽 시프트 연산자 : 왼쪽으로 시프트할 때마다 2^n을 곱한 효과
- 오른쪽 시프트 연산자

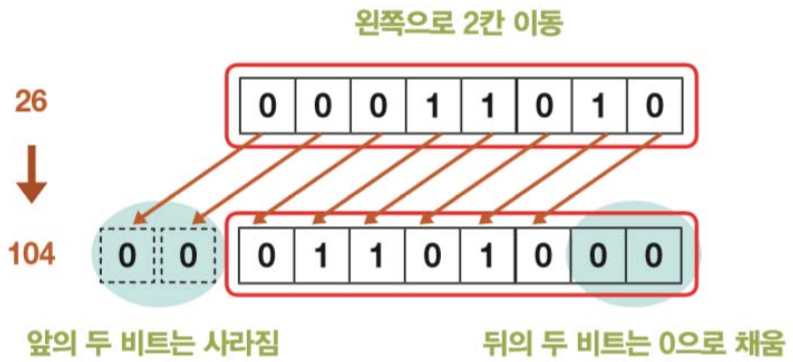


그림 4-7 26의 왼쪽으로 2칸 시프트 연산

그림 4-8 26의 오른쪽으로 2칸 시프트 연산

```
a = 10
a << 1; a << 2; a << 3; a << 4
```

출력 결과

```
20 40 80 160
```

```
a = 10
a >> 1; a >> 2; a >> 3; a >> 4
```

출력 결과

```
5 2 1 0
```

Code04-04.py

```
1  a = 100
2  result = 0
3  i = 0
4
5  for i in range(1, 5) :
6      result = a << i
7      print("%d << %d = %d" % (a, i, result))
8
9  for i in range(1, 5) :
10     result = a >> i
11     print("%d >> %d = %d" % (a, i, result))
```

5행 : for 문은 반복을 위한 것

6~7행 : 4회(i값이 1부터 4까지 변함) 반복

9~11행은 $100//2^1=50$, $100//2^2=25$… 등이 출력

출력 결과

```
100 << 1 = 200
100 << 2 = 400
100 << 3 = 800
100 << 4 = 1600
100 >> 1 = 50
100 >> 2 = 25
100 >> 3 = 12
100 >> 4 = 6
```

Section 05 연산자 우선순위

- 연산자 우선순위 : 여러 개의 연산자가 있을 경우 정해진 순서

연산자	설명
**	지수 연산자
~ , + , -	단항 연산자
* , / , % , //	곱셈, 나눗셈, 나머지 연산자
+ , -	덧셈, 뺄셈
>> , <<	비트 이동 연산자
&	비트 AND 연산자
^ , |	비트 XOR 연산자, 비트 OR 연산자
<= , < , > , >=	비교 연산자
== , !=	동등 연산자
= , %= , /= , //= , -= , += , *= , **=	할당 연산자, 복합 할당 연산자
is , is not	아이덴티티 연산자
in , not in	소속 연산자
not , or , and	논리 연산자

LAB3-5 평균 구하기 – 연산자 우선순위

수식을 프로그램으로 구현하는 경우에는 연산자의 우선순위에 주의할 필요가 있다. 사용자로부터 3개의 수를 입력받아서 평균값을 계산하여 출력하는 프로그램을 살펴보자. 이것을 다음과 같이 코딩하면 안 된다.

```
x = int(input("첫 번째 수를 입력하시오: "))
y = int(input("두 번째 수를 입력하시오: "))
z = int(input("세 번째 수를 입력하시오: "))

avg = x + y + z / 3     # 잘못된 코딩
print("평균 =", avg)
```

이렇게 코딩을 하면 10, 20, 30을 입력했을 때에 평균이 옳은 값인 20이 나오지 않고 40으로 잘못 계산되어 나온다. 우리가 원하는 결과는 아래와 같다. 어디를 고쳐야 할지 생각해 보고 수정해서 결과를 확인하라.

원하는 결과

```
첫 번째 수를 입력하시오: 10
두 번째 수를 입력하시오: 20
세 번째 수를 입력하시오: 30
평균 = 20.0
```

```
x = int(input("첫 번째 수를 입력하시오: "))
y = int(input("두 번째 수를 입력하시오: "))
z = int(input("세 번째 수를 입력하시오: "))

avg = (x + y + z) / 3    # 올바르게 고쳐진 계산
print("평균 =", avg)
```

Lab

잠깐 – 사용자로부터 여러 개의 수를 입력받자.

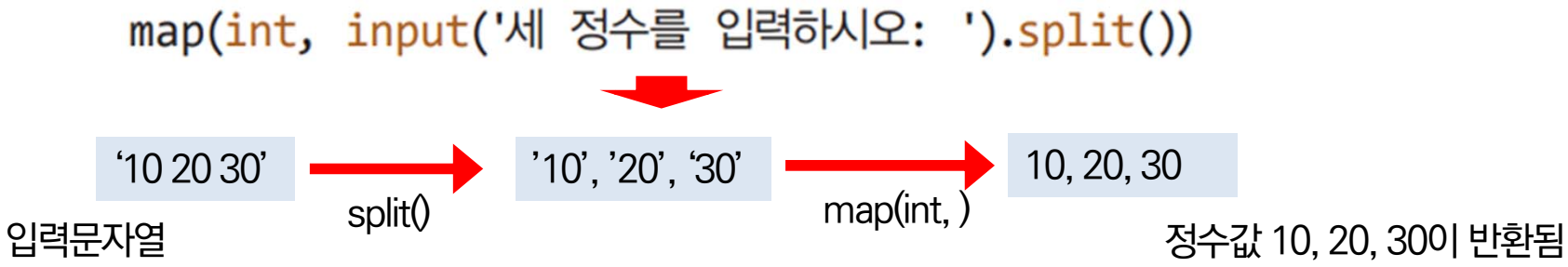
사용자로 부터 개별적인 수를 여러줄에 걸쳐 입력받을 수도 있으나 다음과 같은 코드를 사용하면 한 줄에 3개의 정수를 한꺼번에 입력받을 수 있다.

```
>>> x, y, z = map(int, input('세 정수를 입력하시오: ').split())
세 정수를 입력하시오: 10 20 30
>>> x, y, z
(10, 20, 30)
```

split() 메소드는 입력된 문자열을 공백단위로 나누어서 '10' 과 '20', '30'의 세 문자열로 나누어 주며 이를 map() 함수가 int형으로 바꾸어 주는 일을 한다.

[memo]
map(function, iterable)
두 번째 인자로 들어온 반복 가능한 자료형 (리스트나 튜플)을 첫 번째 인자로 들어온 함수에 하나씩 집어 넣어서 함수를 수행

iterable : 반복 가능한 객체(리스트, 튜플 등)



Section 06 random 모듈과 math 모듈

➢ 난수와 관련한 함수를 제공하는 random 모듈과 수학 함수를 제공하는 math 모듈에 대하여 간단하게 설명한다.

- math 모듈은 원주율 π, 자연상수 e 등의 상수 값과 실수의 절대값 fabs(), ceil(), floor(), log(), sin(), cos() 등의 다양한 수학 함수를 포함하고 있으며 다음과 같이 사용할 수 있다.

```
>>> import math
>>> math.pow(3, 3)       # 3의 3 제곱
27.0
>>> math.fabs(-99)       # -99의 실수 절대값
99.0
>>> math.log(2.71828)
0.999999327347282
>>> math.log(100, 10)    # 로그 10을 밑으로 하는 100값
2
>>> math.pi              # 원주율
3.141592653589793
>>> math.sin(math.pi / 2.0)      # sin() 함수의 인자로 PI/2.0를 넣어보자
1.0
```

- random 모듈은 임의의 수를 생성하거나, 리스트나 튜플내의 요소를 무작위로 섞거나, 선택하는 함수를 포함하고 있다. random 모듈에 포함되어 있는 대표적인 함수 몇 가지를 함께 알아보자.

```
>>> import random
>>> random.random()            # 0 이상 1 미만의 임의의 실수를 반환
0.19452357419514088
>>> random.random()            # 이 함수는 매번 다른 실수를 반환
0.6947454047320903
>>> random.randint(1, 7)       # 1 이상 7 이하(7을 포함)의 임의의 정수를 반환
3
>>> random.randrange(7)        # 0 이상 7 미만(7을 포함하지 않음)의 임의의 정수를 반환
3
>>> random.randrange(1, 7)     # 1 이상 7 미만(7을 포함안함)의 임의의 정수를 반환
6
>>> random.randrange(0, 10, 2)  # 0, 2, 4, 8 중(10은 포함 안함) 하나를 반환함
2
>>> lst = [10, 20, 30, 40, 50]  # 여러개의 값을 가지는 리스트를 생성함
>>> random.shuffle(lst)         # lst 리스트의 순서를 무작위로 섞음
>>> lst
[40, 50, 10, 20, 30]
>>> random.choice(lst)          # lst 리스트의 원소 중 무작위로 하나를 고름
30
```