

## 연산자

## 학습목표

- 더하기, 빼기, 곱하기, 나누기 등 산술 연산자의 사용법을 익힌다.
- 관계 · 논리 · 비트 연산자의 사용법을 익힌다.
- 연산자 우선순위를 익힌다.
- 산술 연산자를 이용해 동전 교환하는 프로그램을 만든다.
- 논리 연산자를 이용해 거북이가 마음대로 이동하는 프로그램을 만든다.

SECTION 01 이 장에서 만들 프로그램

SECTION 02 산술 연산자

SECTION 03 관계 연산자

SECTION 04 논리 연산자

SECTION 05 비트 연산자

SECTION 06 연산자 우선순위

요약

연습문제

응용예제



- 파이썬 3.6부터 .format 대신 f-string 사용 가능

```
a = 'a'  
print(f'a is {a}')
```

```
x, y, z = 1, 2, 3  
print(f'a is {x}, {y}, {z}')
```

```
print(f'a is {z}, {y}, {x}')
```

```
name = 'Steve'  
family = 'Jobs'  
print(f'My name is {name} {family}.  
I am {family} {name}')
```

## ■ 실습 문제

- f-string 코드를 .format 코드로 변경하시오.

```
a = 'a'
print(f'a is {a}')
```

```
x, y, z = 1, 2, 3
print(f'a is {x}, {y}, {z}')
```

```
print(f'a is {z}, {y}, {x}')
```

```
name = 'Jun'
family = 'Sakai'
print(f'My name is {name} {family}.
I am {family} {name}')
```

# 한 줄이 길어질 경우

```
s1 = 'aaaaaaaaaaa' \
    + 'bbbbbbbbbbbbbbbbbb'
print(s1)

s2 = ('aaaaaaaaaaa'
    + 'bbbbbbbbbbbbbbbbbb')
print(s2)

x1 = 1 + 1 + 1 + 1 + 1 + 1 + 1 \
    + 1 + 1 + 1 + 1 + 1 + 1 + 1
print(x1)

x2 = (1 + 1 + 1 + 1 + 1 + 1 + 1
    + 1 + 1 + 1 + 1 + 1 + 1 +
1)
print(x2)
```

output :

```
aaaaaaaaaabbbbbbbbbbbbbbbb
aaaaaaaaaabbbbbbbbbbbbbbbb
14
14
```

# 출력 결과는?

```
print("hello")  
print('\hello')
```

```
print("I don't know")  
print('I don\t know')
```

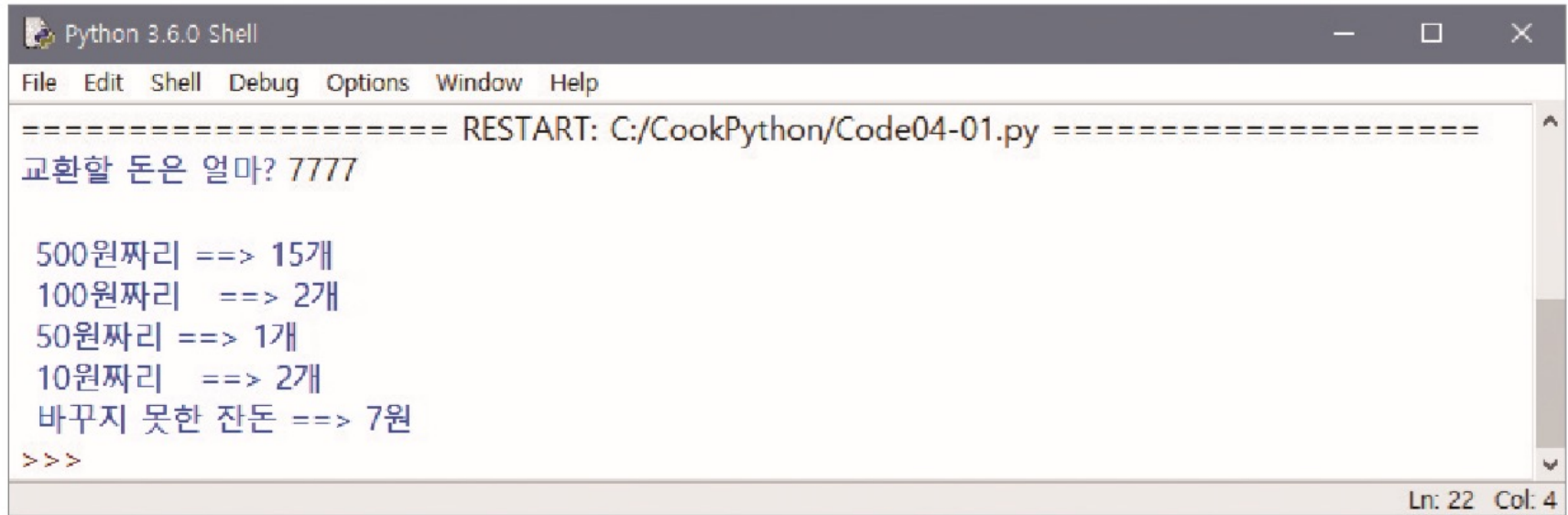
```
print(r'c:\usr\bin')
```

```
print("""\  
line1  
line2  
line3\  
""")
```

```
print("""aaaaaaaaaaaaaaaaa  
bbbbbbbbbbbbbbbbbb  
""")
```

# Section 01 이 장에서 만들 프로그램

## ■ [프로그램 1] 동전교환



```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/CookPython/Code04-01.py =====
교환할 돈은 얼마? 7777

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

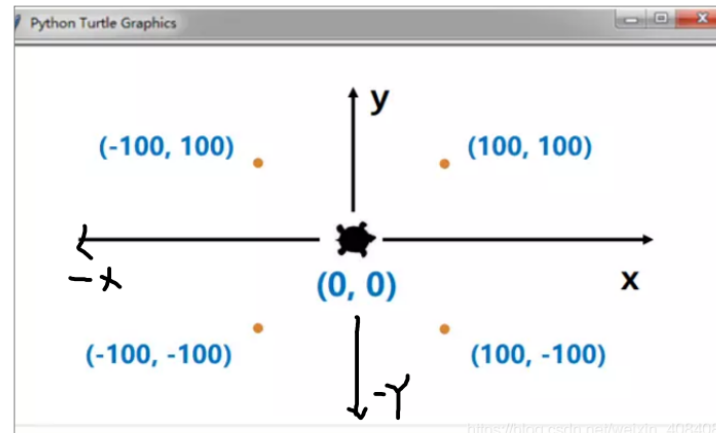
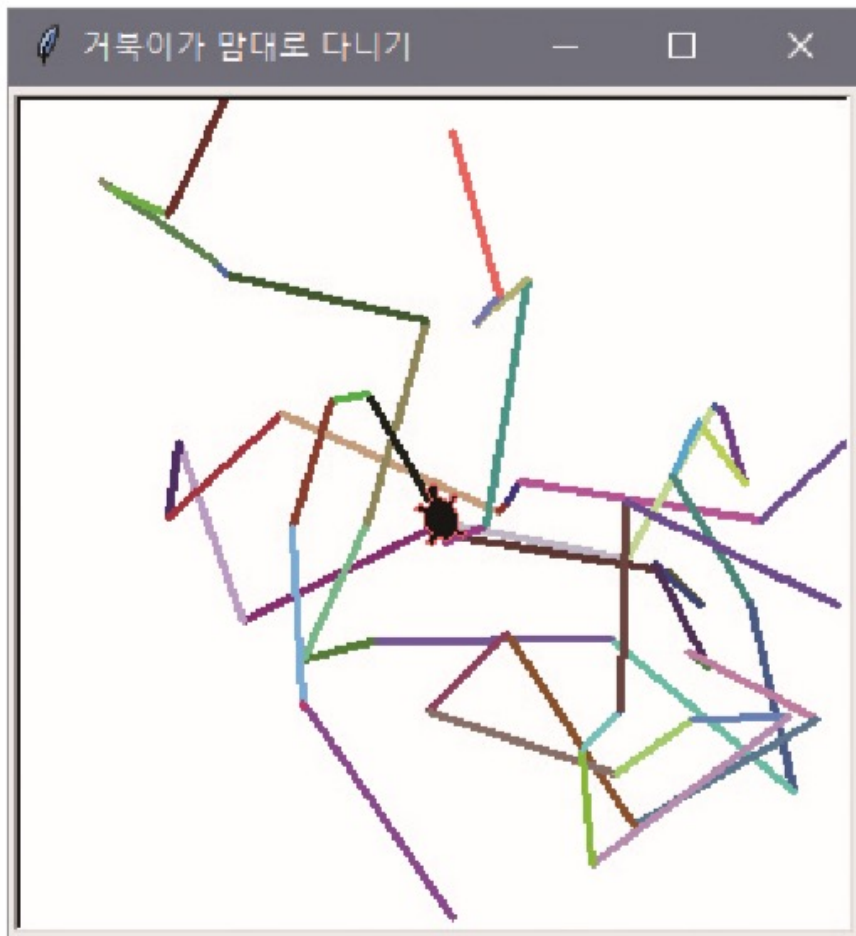
Ln: 22 Col: 4

$7777 \div 500 = 5 \rightarrow 277$   
 $277 \div 100 = 2 \rightarrow 77$   
 $77 \div 50 = 1 \rightarrow 27$   
 $27 \div 10 = 2 \rightarrow 7$   
연산자 // : 몫      % : 나머지

## Section 01 이 장에서 만들 프로그램

### ■ [프로그램 2] 마음대로 이동하는 거북이

- 거북이가 화 면 안에서 마음대로 이동하게 하는 프로그램.
- 단, 거북이가 화면을 벗어날 때는 다시 화면의 중앙으로 옮긴 후 마음대로 이동하도록 설정



100 + 100

100 + 100

## Section 02 산술 연산자

### ■ 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



## Section 02 산술 연산자

- $a//b$ 는  $a$ 를  $b$ 로 나눈 몫이고,  $a\%b$ 는  $a$ 를  $b$ 로 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  $a=5; b=3$ 은 다음과 동일하다.  
또 콤마(,)로 분리해서 값을 대입할 수도 있어  $a, b=5, 3$ 도 동일

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

## Section 02 산술 연산자

### ■ 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  $a + (b - c)$

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

## Section 02 산술 연산자

### ■ 산술 연산자의 우선순위

❶  $(a + b) * c \rightarrow (2 + 3) * 4 \rightarrow 5 * 4 \rightarrow 20$

❷  $a + (b * c) \rightarrow 2 + (3 * 4) \rightarrow 2 + 12 \rightarrow 14$

- 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

[illegible]

## 출력 결과

101 101.123 10000000000000000000000000000000

## Section 02 산술 연산자

- 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

```
>>> a = 100
```

```
>>> b = 100.123
```

```
>>> str(a)
```

```
'100'
```

```
>>> print(a)
```

```
100
```

### ■ 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

## Section 02 산술 연산자

- 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

## Section 02 산술 연산자

### ■ [프로그램 1]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 저장 할 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

몫

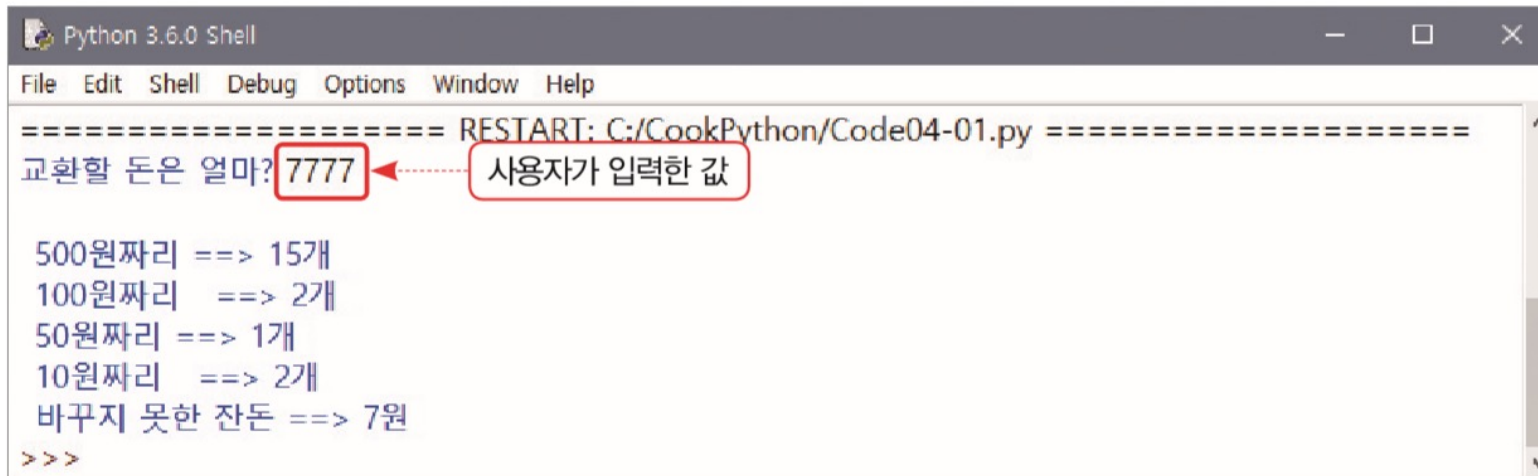
4머지



## Section 02 산술 연산자

```
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

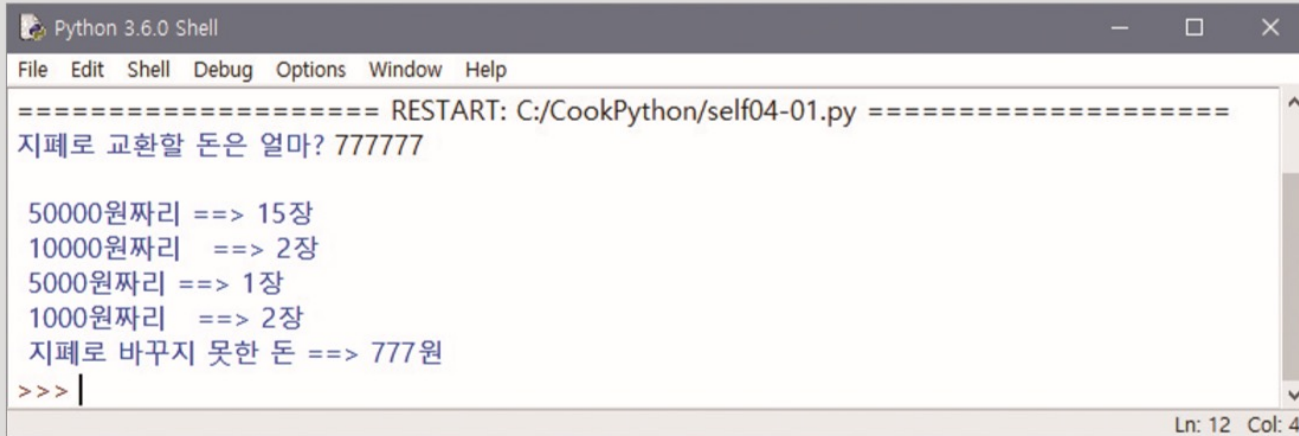


```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/CookPython/Code04-01.py =====
교환할 돈은 얼마? 7777
500원짜리 ==> 15개
100원짜리  ==> 2개
50원짜리   ==> 1개
10원짜리   ==> 2개
바꾸지 못한 잔돈 ==> 7원
>>>
```

## Section 02 산술 연산자

### SELF STUDY 4-1

돈(예로 777777)을 입력하면 5만 원, 1만 원, 5000원, 1000원 지폐로 교환하는 프로그램을 작성해 보자.



```
Python 3.6.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/CookPython/self04-01.py =====
지폐로 교환할 돈은 얼마? 777777

50000원짜리 ==> 15장
10000원짜리 ==> 2장
5000원짜리 ==> 1장
1000원짜리 ==> 2장
지폐로 바꾸지 못한 돈 ==> 777원
>>> |
```

Ln: 12 Col: 4

### ■ 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

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

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

## Section 03 관계 연산자

- `a==b`를 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=b`는 b 값을 a에 대입하라는 의미이지 관계 연산자가 아님

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

 오류

## Section 04 논리 연산자

### ■ 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	<code>(a &gt; 100) and (a &lt; 200)</code>
or(논리합)	~이거나, 또는	둘 중 하나만 참이어도 참	<code>(a == 100) or (a == 200)</code>
not(논리부정)	~아니다, 부정	참이면 거짓, 거짓이면 참	<code>not(a &lt; 100)</code>

## Section 04 논리 연산자

```
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 모두 True

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

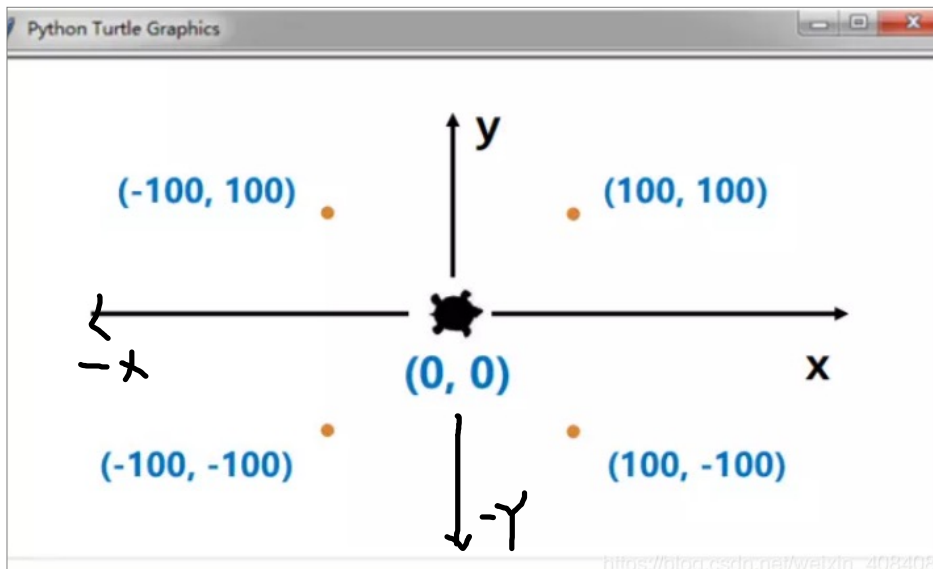
## Section 04 논리 연산자

### ■ [프로그램 2]의 완성

- 마음대로 이동하는 거북이 프로그램 구현

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

### ■ Turtle Screen Coordinate



$100 + 100$

$100 + 100$

## Section 04 논리 연산자


### ■ [프로그램 2]의 완성

- 마음대로 이동하는 거북이 프로그램 구현

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 윈도우창의 폭과 높이, pSize는 펜의 두께 준비 또 exitCount는 윈도우창 밖으로 빠져나간 횟수를 위해서 준비. r, g, b는 색상, angle과 dist는 임의로 이동할 거리와 각도, curX와 curY는 현재 거북이의 위치를 지정하는 변수



0, 0, 0, 0, 0, 0, 0



## Section 04 논리 연산자

```
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     angle = random.randrange(0, 360)
21     dist = random.randrange(1, 100)
22     turtle.left(angle)
23     turtle.forward(dist)
24     curX = turtle.xcor()
```

9~13행 : 창의 제목, 거북이 모양, 펜 두께, 윈도우 크기, 안쪽 화면 크기 지정

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

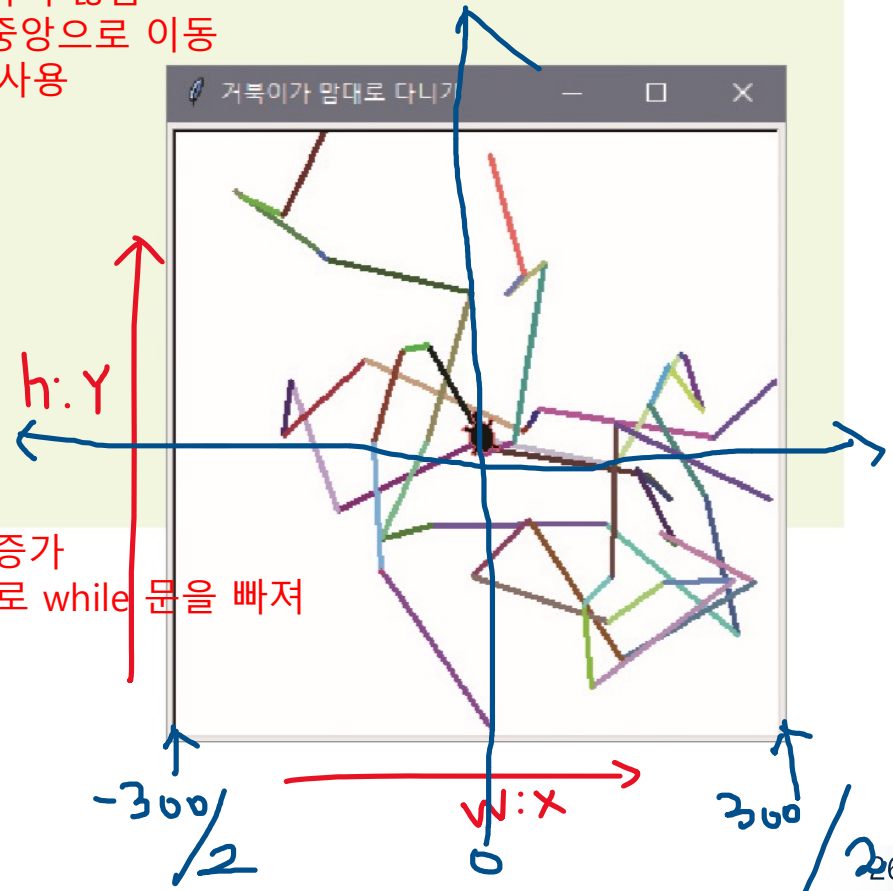
## Section 04 논리 연산자

```
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 빠져  
나간 후 프로그램 종료



### ■ 비트 연산자의 개념

- 정수를 2진수로 변환한 후 각 자리의 비트끼리 연산 수행
- 비트 연산자의 종류 :  $\&$ ,  $|$ ,  $\wedge$ ,  $\sim$ ,  $\ll$ ,  $\gg$

표 4-5 비트 연산자의 종류

연산자	의미	설명
$\&$	비트 논리곱(and)	둘 다 1이면 1
$ $	비트 논리합(or)	둘 중 하나만 1이면 1
$\wedge$	비트 논리적 배타합(xor)	둘이 같으면 0, 다르면 1
$\sim$	비트 부정	1은 0으로, 0은 1로 변경
$\ll$	비트 이동(왼쪽)	비트를 왼쪽으로 시프트(Shift)
$\gg$	비트 이동(오른쪽)	비트를 오른쪽으로 시프트(Shift)

## Section 05 비트 연산자

- $123 \& 456$ 은 123의 2진수인 11110112와 456의 2진수인 1110010002의 비트 논리곱(&) 결과인 10010002가 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

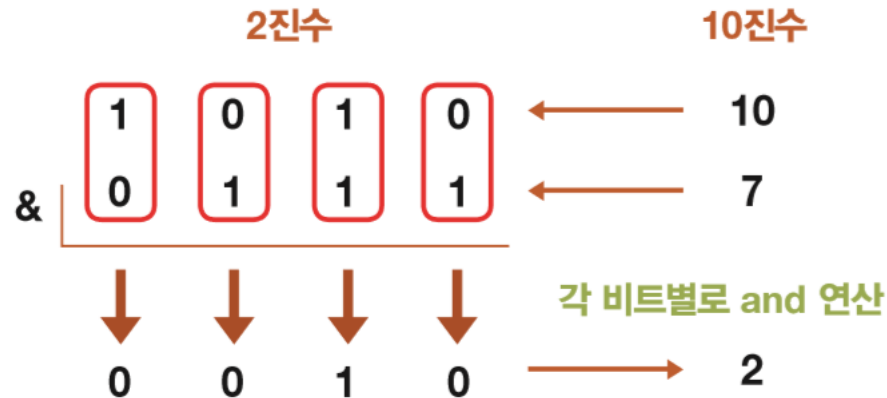


그림 4-2 비트 논리곱의 예

## Section 05 비트 연산자

- 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

## Section 05 비트 연산자

- 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

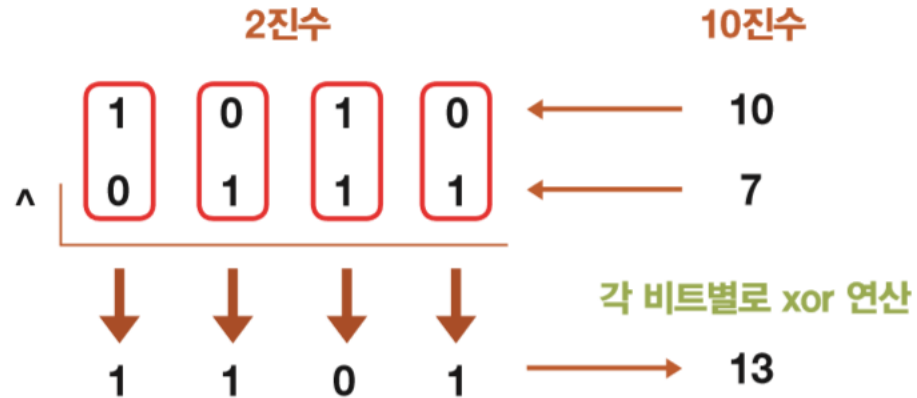


그림 4-4 비트 배타적 논리합의 예

$10 \wedge 7$

$123 \wedge 456$

$0xFFFF \wedge 0x0000$

출력 결과

13 435 65535

## Section 05 비트 연산자

### ■ 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
```



## Section 05 비트 연산자

- 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

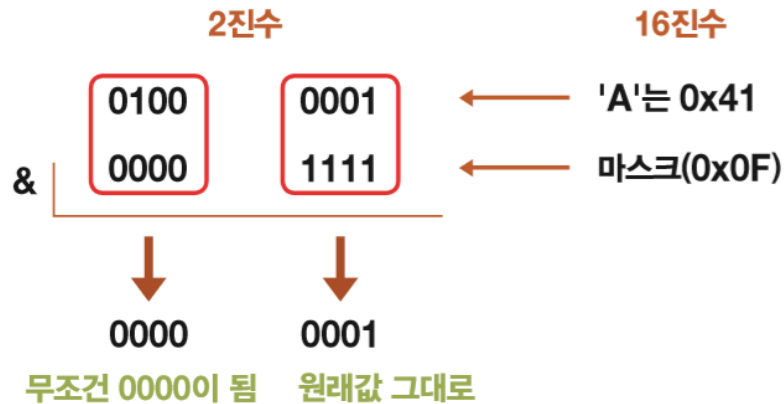


그림 4-5 마스크 0x0F를 사용한 비트 논리곱 결과



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 0x41이므로 두 값의 차이는 0x20이 됨. 이는 10진수로 32이고, 2진수로는 0010 0000. 9행에서 A와 32(0010 00002)의 비트 배타적 논리합 수행하면 a로 변경, 11행에서 다시 a와 32(0010 00002)의 비트 배타적 논리합 수행하면 A로 원상 복귀

## Section 05 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
```

## Section 05 비트 연산자

### ■ 시프트 연산자

- 왼쪽 시프트 연산자 : 왼쪽으로 시프트할 때마다  $2^n$ 을 곱한 효과

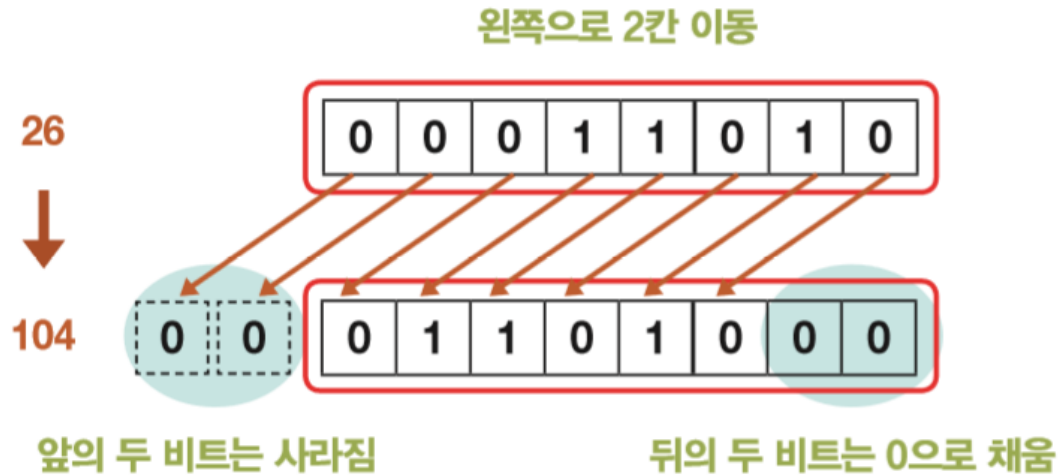


그림 4-7 26의 왼쪽으로 2칸 시프트 연산

`a = 10`

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

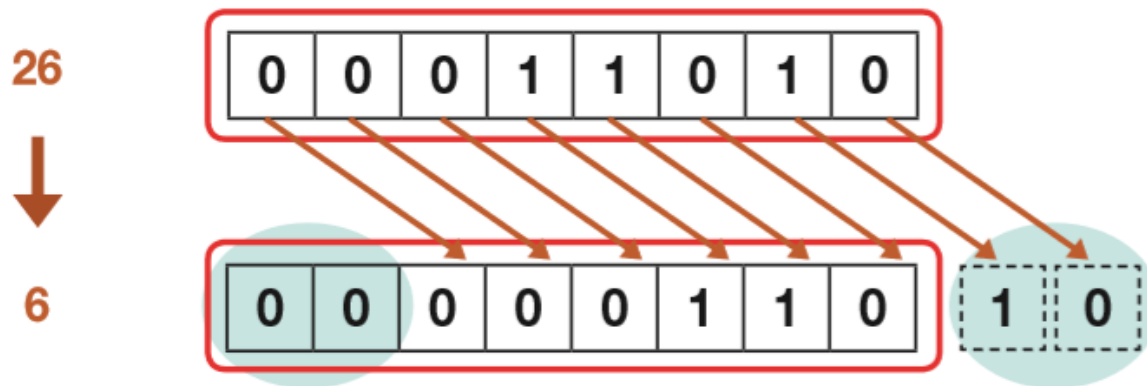
출력 결과

20 40 80 160

## Section 05 비트 연산자

### 오른쪽 시프트 연산자

오른쪽으로 2칸 이동



앞의 두 비트는 부호 비트(0)로 채움

뒤의 두 비트는 사라짐

그림 4-8 26의 오른쪽으로 2칸 시프트 연산

```
a = 10
```

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

출력 결과

```
5 2 1 0
```

## Section 05 비트 연산자

Code04-04.py

```
1  a = 100
2  result = 0
3  i = 0
4
5  for i in range(1, 5) :    5행 : for 문은 반복을 위한 것
6      result = a << i      6~7행 : 4회(i값이 1부터 4까지 변함) 반복
7      print("%d << %d = %d" % (a, i, result))
8
9  for i in range(1, 5) :
10     result = a >> i      9~11행은 100//21=50, 100//22=25... 등이 출력
11     print("%d >> %d = %d" % (a, i, result))
```

## Section 05 비트 연산자

### 출력 결과

$100 \ll 1 = 200$

$100 \ll 2 = 400$

$100 \ll 3 = 800$

$100 \ll 4 = 1600$

$100 \gg 1 = 50$

$100 \gg 2 = 25$

$100 \gg 3 = 12$

$100 \gg 4 = 6$

## Section 06 연산자 우선순위

### ■ 연산자 우선순위 : 여러 개의 연산자가 있을 경우 정해진 순서

표 4-6 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



**Thank You**

---