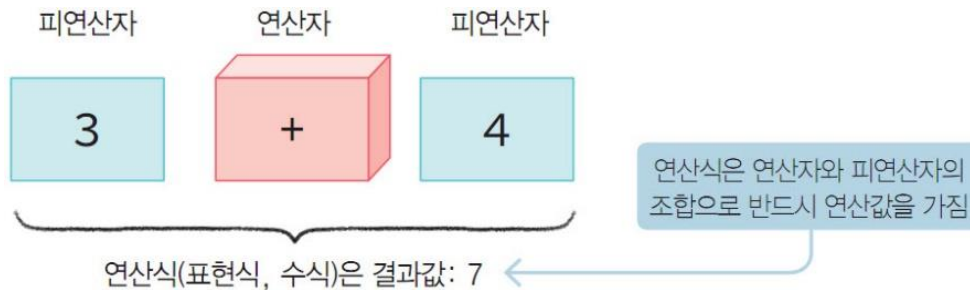


연산자

연산자

❖ 식

- 계산을 표현하는 기본적인 수단으로 연산자, 피연산자, 괄호, 함수 호출 등으로 구성
- 피연산자 : 연산에 참여하는 변수나 값
- 연산자 : 연산을 수행하기 위한 기호나 키워드
- 예) $3+4$
 - ✓ '+'는 연산자이고, 3과 4는 피연산자
 - ✓ 항상 하나의 결과값: 7



연산자

❖ 피연산자의 개수에 따른 연산자의 종류

- 한 개의 피연산자를 갖는 단항 연산자
- 두 개의 피연산자를 갖는 이항 연산자

연산자의 종류	의미	연산자
단항 연산자	피연산자가 한 개인 경우	+, -, ++, --, !, &, ~, sizeof
이항 연산자	피연산자가 두 개인 경우	+, -, *, /, %, =, >, <, >=, <=, ==, !=, &&, , &, , ^, <<, >>, +=, -=, *=, /=, %=, >>=, <<=, &=, =, ^=
삼항 연산자	피연산자가 세 개인 경우	?:

연산자

❖ 연산자의 기능에 따른 연산자의 종류

연산자의 종류	연산자
산술 연산자	+, -, *, /, %
증감 연산자	++, --
관계 연산자	>, <, >=, <=, ==, !=
논리 연산자	&&, , !
비트 연산자	&, , ^, ~, <<, >>
대입 연산자	=, +=, -=, *=, /=, %=, &=, =, ^=, >>=, <<=
조건 연산자	?:
그 밖의 연산자	,(콤마 연산자), sizeof, 형 변환 연산자

산술 연산자

❖ 산술 연산자

➤ 수학적 계산을 위해 사용하는 연산자

연산자	의미	사용 예	설명
+	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

산술 연산자

❖ 산술 연산자 사용 예

- $a//b$ 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
```

- 세미콜론(;)은 앞뒤를 완전히 분리
 - ✓ $a=5; b=3$ 은 다음과 동일

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

- 콤마(,)로 분리해서 값을 대입할 수도 있어 $a, b=5, 3$ 도 동일

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

➤ $a+b-c$

- ✓ 덧셈과 뺄셈은 연산자 우선순위가 동일하므로 어떤 것을 먼저 계산하든 동일

❶ $(a + b) - c$

❷ $a + (b - c)$

- ✓ 특별히 괄호가 없을 때는 왼쪽에서 오른쪽 방향으로 계산(1번과 같이 계산)

산술 연산자

❖ $a+b*c$ 예

$$\textcircled{1} (a + b) * c \rightarrow (2 + 3) * 4 \rightarrow 5 * 4 \rightarrow 20$$

$$\textcircled{2} a + (b * c) \rightarrow 2 + (3 * 4) \rightarrow 2 + 12 \rightarrow 14$$

- 덧셈(또는 뺄셈)과 곱셈(또는 나눗셈)이 같이 있으면 곱셈(또는 나눗셈)이 먼저 계산된 후 덧셈(또는 뺄셈)이 계산
- 괄호가 없어도 **②**처럼 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 계산
- 괄호를 사용하면 무조건 괄호가 우선 계산

$$\textcircled{1} a = b + c * d;$$

$$\textcircled{2} a = b + (c * d);$$

산술 연산자

❖ 산술 연산을 하는 문자열과 숫자의 상호 변환

- 숫자로 되어 있지만 수학적 계산용이 아닌 경우 문자 또는 문자열로 사용
 - ✓ 문자열은 다수의 문자로 구성된 것으로 것을 의미
- 문자열이 숫자로 구성되어 있을 때 정수나 실수로 변환하는 함수가 제공
- 문자열은 `int()` 함수를 통해 정수로 `float()` 함수를 통해 실수로 변경

[illegible]

출력 결과

101 101.123 10000000000000000000000000000000

산술 연산자

❖ 숫자를 문자열로 변환

- 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'
```

산술 연산자

❖ 산술 연산자와 대입 연산자

- 대입 연산자(=)는 연산자의 좌변(변수)에 우변의 값을 저장
- 대입 연산자의 좌변(l-value)에는 반드시 변수만 사용
- 대입 연산자 = 외에도 산술연산자와 함께 사용하는 대입연산자가 제공
 - ✓ +=, -=, *=, /=, %=, //=, **=
- 파이썬에는 증가 연산자 ++나 감소 연산자 --가 없음

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

산술 연산자

❖ 값의 누적 예

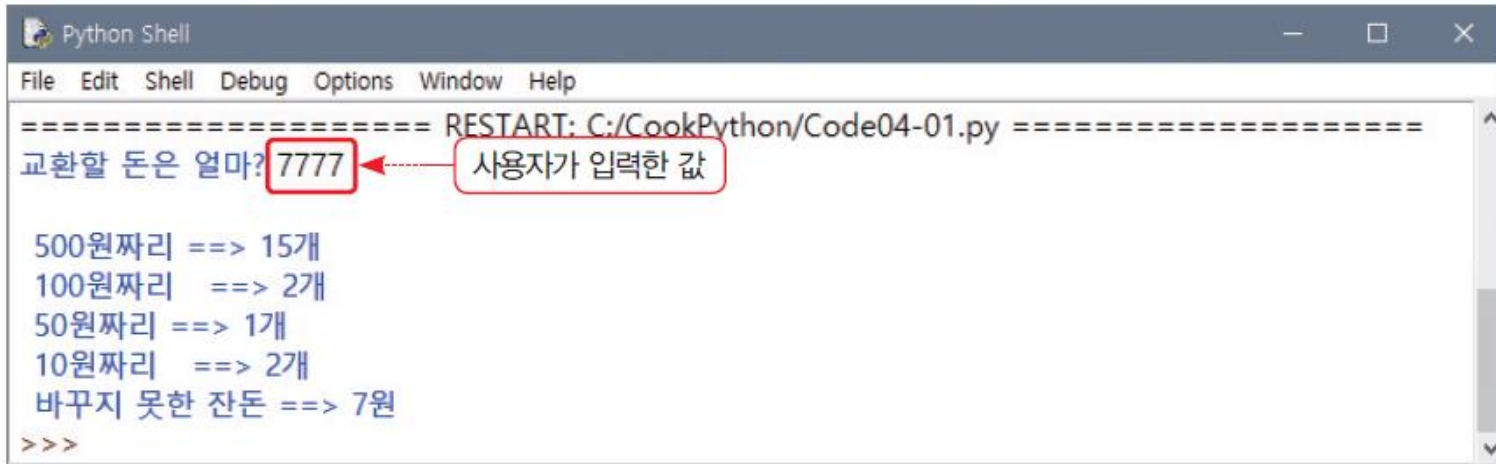
```
a = 10  
a += 5; print(a)  
a -= 5; print(a)  
a *= 5; print(a)  
a /= 5; print(a)  
a //= 5; print(a)  
a %= 5; print(a)  
a **= 5; print(a)
```

출력 결과

15 10 50 10.0 2.0 2.0 32.0

산술 연산자

❖ 학습한 연산자를 활용해서 동전 교환 프로그램 구현(1/3)



```
Python 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
>>>
```

산술 연산자

❖ 학습한 연산자를 활용해서 동전 교환 프로그램 구현(2/3)

```
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

산술 연산자

❖ 학습한 연산자를 활용해서 동전 교환 프로그램 구현(3/3)

```
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

관계 연산자

❖ 관계연산자

- 어떤 것이 크거나 작거나 같은지 비교하는 것
 - ✓ 참은 True로, 거짓은 False로 표시

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

- 조건문(if)이나 반복문(while)에서 사용하며 단독으로는 거의 사용하지 않음

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

관계 연산자

❖ 관계 연산자 사용 예

- `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)
```

논리 연산자

❖ 논리 연산자

- 여러 가지 조건을 복합해서 조건식이나 값을 참과 거짓을 판별
- 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)`

연산자	의미	설명	사용 예
and(논리곱)	~이고, 그리고	둘 다 참이어야 참	<code>(a > 100) and (a < 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 < 100)</code>

논리 연산자

❖ 논리 연산자 사용 예

```
a = 99
```

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

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

```
not(a == 100)
```

출력 결과

```
False True True
```

논리 연산자

❖ 숫자도 참과 거짓으로 구분

- 숫자 0은 거짓이고 그외 값은 모두 참으로 취급
- 예) 조건문에서 사용

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

- ✓ 첫 번째 1234는 참으로 취급하므로 결과 출력
- ✓ 두 번째 0은 거짓이므로 결과가 출력되지 않음

비트 연산자

❖ 비트 연산자

- 컴퓨터 내부적으로는 모든 숫자와 문자를 비트로 변환하여 저장(코드)
- 정수를 2진수로 변환한 후 각 자리의 비트끼리 연산 수행
- 비트 연산자의 종류 : $\&$, $|$, \wedge , \sim , \ll , \gg

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)

비트 연산자

❖ 비트 연산자 사용 예

10 & 7

123 & 456

0xFFFF & 0x0000

출력 결과

2 72 0

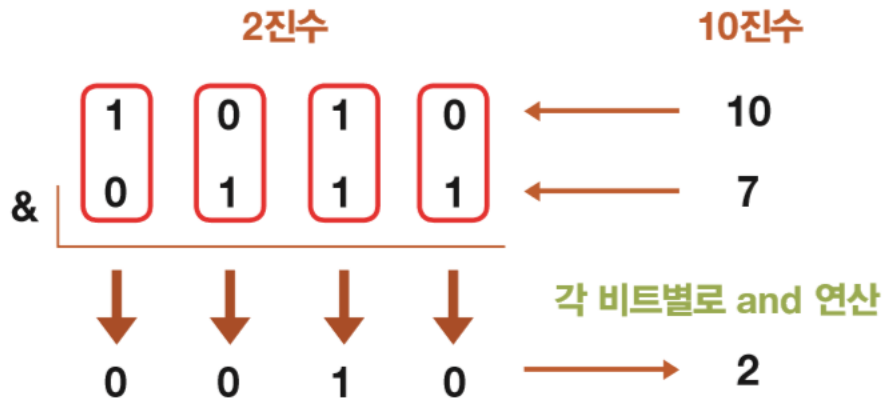
- $123 \& 456$ 은 123의 2진수인 1111011_2 와 456의 2진수인 111001000_2 의 비트 논리곱(&) 결과인 1001000_2 가 되므로 10진수로 72가 나옴
- 두 수의 자릿수가 다를 때는 빈 자리에 0을 채운 후 비트 논리곱 연산
- 0과 비트 논리곱을 수행하면 어떤 숫자든 무조건 0

비트 연산자

❖ 비트 논리곱

- &는 비트 논리곱을 수행한 결과가 나옴
- 비트 연산은 0과 1밖에 없으므로 0은 False, 1은 True
- 10&7의 결과는 2

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



비트 연산자

❖ 비트 논리합

➤ $10 \mid 7$ 의 결과는 15

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

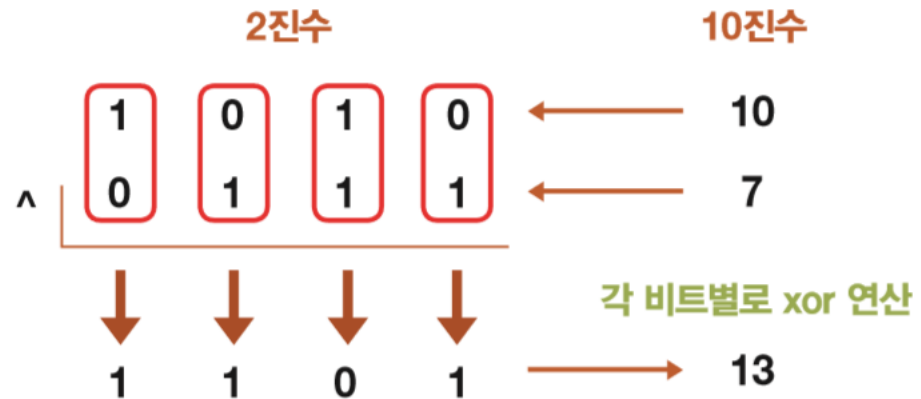


비트 연산자

❖ 비트 배타적 논리합

- 두 값이 다르면 1, 같으면 0
- $10 \wedge 7$ 의 결과는 13

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



비트 연산자

❖ 다수의 입력된 신호를 사용하는 예

- 다수의 입력된 값에 대해 비트 연산자를 수행하여 입력된 값을 확인
- 비트 논리곱, 비트 논리합, 비트 배타적 논리합을 통해 출력 결과 계산

1 번째	0	1	0	1	1	0	0
2 번째	1	1	0	0	0	1	0
결과							

비트 연산자

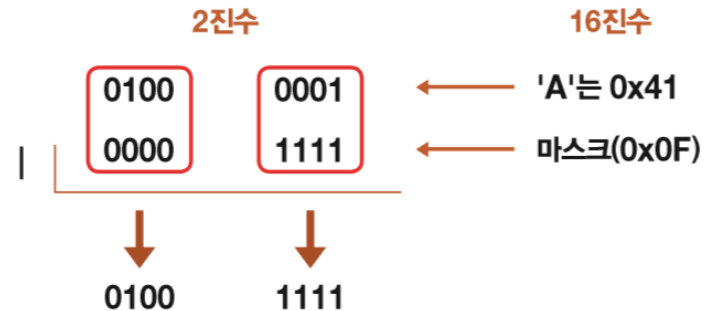
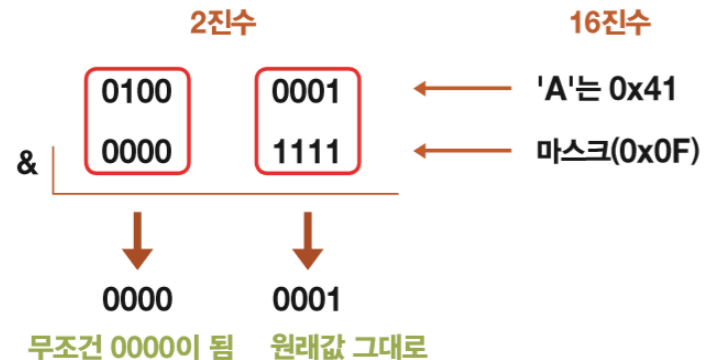
❖ 비트 연산 활용 예

```
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
```

- 마스크(Mask) 방식에 대한 예제(마스크는 무엇을 걸러 주는 역할)
- 우선 마스크값을 2행에서 16진수 0x0F로 선언. 이는 2진수로 0000 1111 의미



비트 연산자

❖ 비트 부정 연산자(또는 보수 연산자)

- 두 수를 연산하는 것이 아니라 하나만 가지고 각 비트를 반대로 만드는 연산
- 반전된 값을 1의 보수라 하고 그 값에 1을 더한 값을 2의 보수
- 해당 값의 음수(-)값을 찾고자 할 때 사용
- 정수값에 비트 부정을 수행한 후 1을 더하면 해당 값의 음수값을 얻는 코드

a = 12345

$\sim a + 1$

출력 결과

-12345

비트 연산자

❖ 시프트 연산자

- 비트로 나열된 값으로 오른쪽이나 왼쪽으로 이동
- 왼쪽 시프트 연산자 : 왼쪽으로 시프트할 때마다 2^n 을 곱한 효과



- 오른쪽 시프트 연산자



비트 연산자

❖ 시프트 연산 예

➤ 왼쪽 시프트 연산자

```
a = 10
```

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

출력 결과

```
20 40 80 160
```

➤ 오른쪽 시프트 연산자

```
a = 10
```

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

출력 결과

```
5 2 1 0
```

연산자 우선순위

❖ 연산자 우선순위

➤ 여러 개의 연산자가 있을 경우 정해진 순서

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