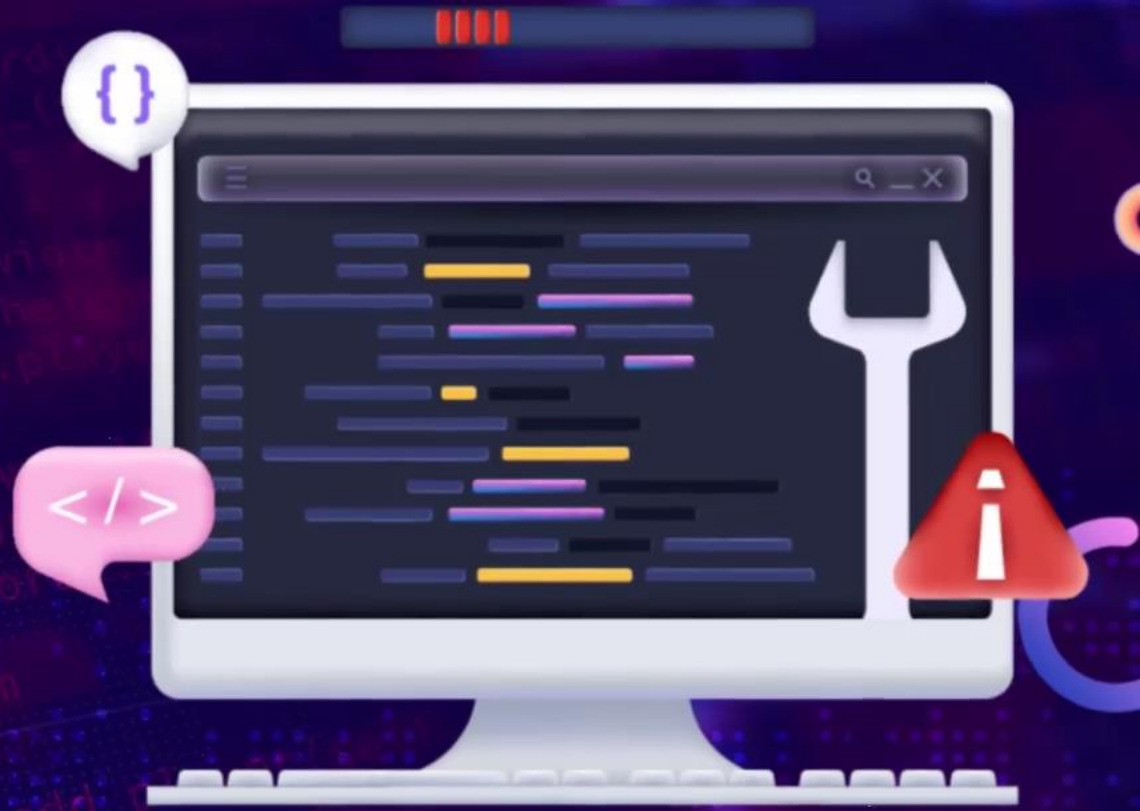
COSS

프로그래밍 기초

—

연산자



비트 연산자 및 연산자 우선순위

- 비트 연산자의 기본 개념을 이해하고 설명할 수 있다.
- 연산자 우선순위를 고려하여 정확한 계산을 수행할 수 있다.

비트 연산자의 개념

❖ 정수를 2진수로 변환한 후 각 자리의 비트끼리 연산 수행

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, 0을 1로 변경
<<	비트 이동(왼쪽)	비트를 왼쪽으로 시프트(Shift)
>>	비트 이동(오른쪽)	비트를 오른쪽으로 시프트(Shift)

비트 연산자의 개념

- 123&456은
  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 숫자이든 무조건 0이 됨

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

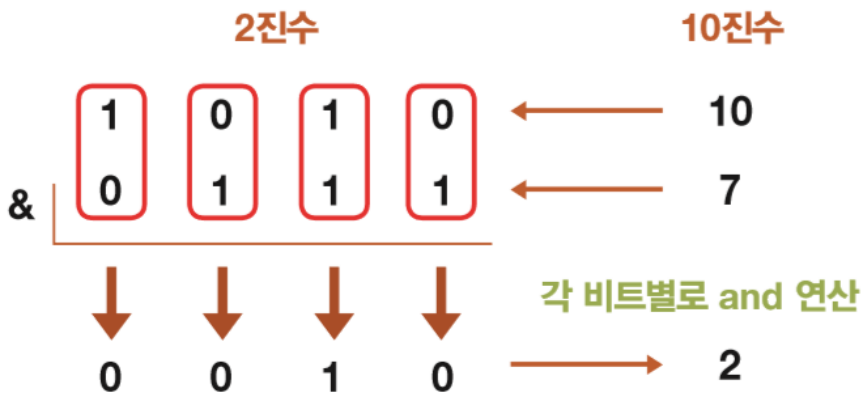
출력 결과

```
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
  - 다음과 같이 10진수를 2진수로 변환한 후
    각 비트마다 and 연산을 수행하기 때문
  - 그 결과 2진수로는 00102가 되고, 10진수로는 2가 됨

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



비트 논리곱과 비트 논리합 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

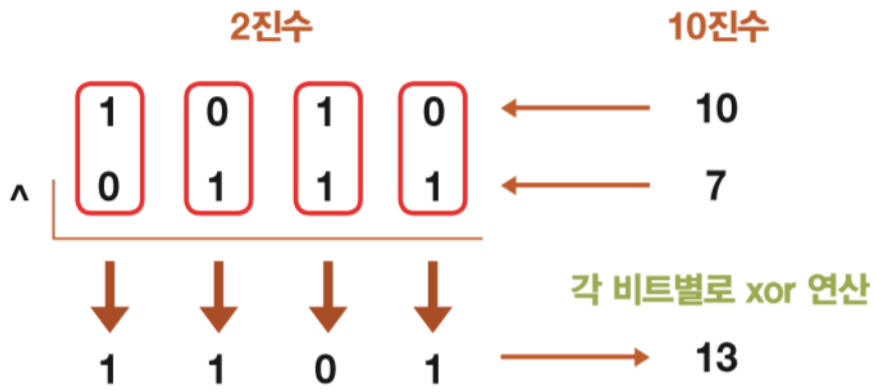
```
15 507 65535
```

비트 논리곱과 비트 논리합 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




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

출력 결과

```
13 435 65535
```

비트 논리곱과 비트 논리합 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
```

비트 논리곱과 비트 논리합 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 보수라고 함
- 해당 값의 음수(-)값을 찾고자 할 때 사용
- 정수값에 비트 부정을 수행한 후 1을 더하면
  해당 값의 음수값을 얻는 코드

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

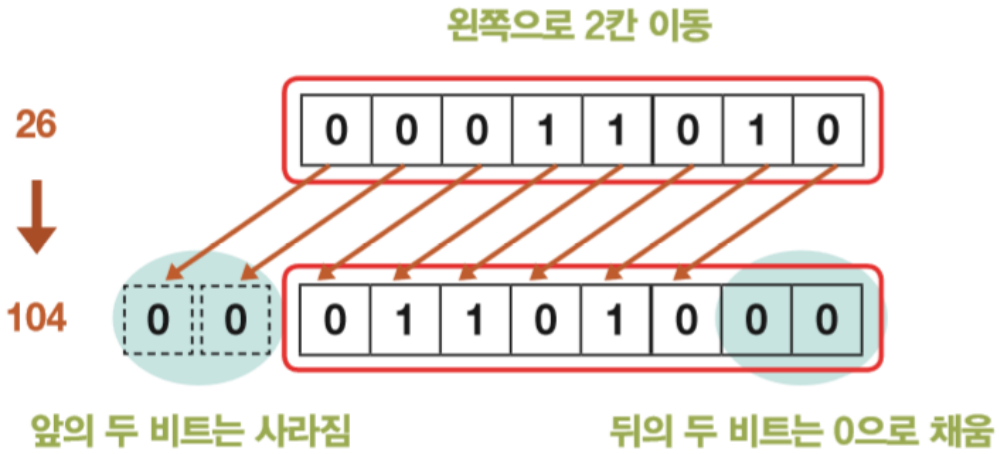
출력 결과

```
-12345
```

시프트 연산자

왼쪽 시프트 연산자

- 왼쪽으로 시프트할 때마다 2^n을 곱한 효과




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

출력 결과

```
20 40 80 160
```

시프트 연산자

오른쪽 시프트 연산자


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

출력 결과

```
5 2 1 0
```

시프트 연산자

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
9~11행	▪ 100//21=50, 100//22=25… 등이 출력

시프트 연산자

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
```

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

연산자 우선순위

❖ 여러 개의 연산자가 있을 경우 정해진 순서

우선순위	연산자	의미
9	|	비트 논리합
10	< > >= <=	관계 연산자
11	== !=	동등 연산자
12	= %= /= //= -= += *= **=	대입 연산자
13	not	논리 연산자
14	and	
15	or	
16	if~else	비교식