

기본 연산자, 조건문 & 반복문

Seolyoung Jeong, Ph.D.

경북대학교 IT대학 컴퓨터학부

기본연산자

연산자

- ◆ 프로그래밍 언어는 일반적으로 수학 연산과 유사한 연산자를 지원함
- ◆ 내장 연산자
 - ◆ 프로그래밍 언어에 따라 내장 연산자는 복잡한 구현을 할 수도 있음
 - ◆ 예) 파이썬에서 '+'는 문자열 연결 가능
- ◆ 피연산자의 위치는 언어에 따라 다를 수 있음
- ◆ 전위/중위/후위 표기법
- ◆ 연산자
 - 산술연산자
 - 관계연산자 (비교연산자)
 - 논리연산자
 - 비트연산자
 - 기타연산자... (포인터 연산자, 복합 연산자 등등)

산술연산자

산술 연산자

◆ 산술 연산자의 종류

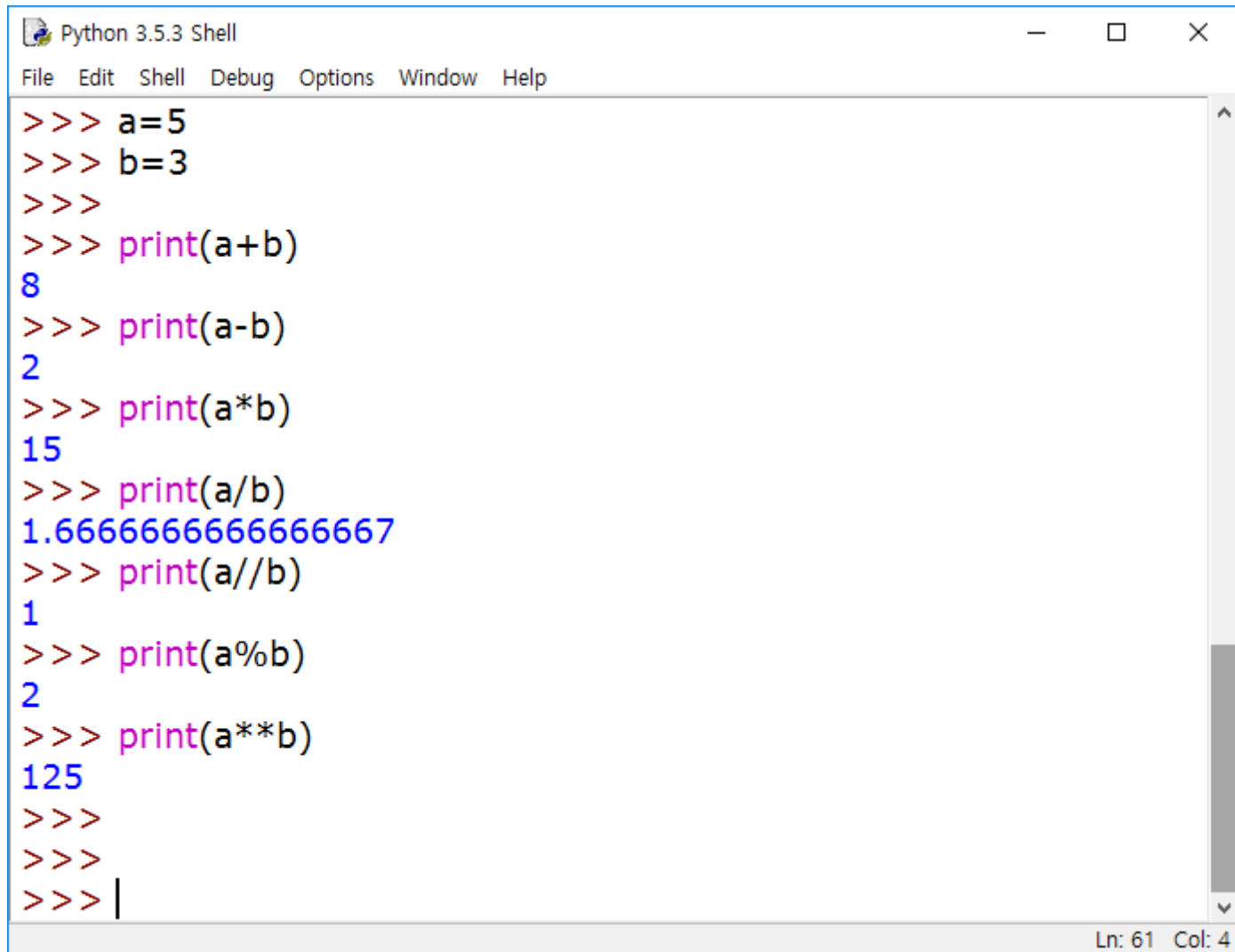
산술 연산자	설명	사용 예	예 설명
=	대입 연산자	$a = 3$	정수 3을 a에 대입
+	더하기	$a = 5 + 3$	5와 3을 더한 값을 a에 대입
-	빼기	$a = 5 - 3$	5에서 3을 뺀 값을 a에 대입
*	곱하기	$a = 5 * 3$	5와 3을 곱한 값을 a에 대입
/	나누기	$a = 5 / 3$	5를 3으로 나눈 값을 a에 대입
//	나누기(몫)	$a = 5 // 3$	5를 3으로 나눈 뒤 소수점을 버리고 a에 대입
%	나머지 값	$a = 5 \% 3$	5를 3으로 나눈 뒤 나머지 값을 a에 대입
**	제곱	$a = 5 ** 3$	5의 3제곱을 a에 대입

```
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
```

산술 연산자 연습 1)



A screenshot of a Python 3.5.3 Shell window. The window has a title bar with the text 'Python 3.5.3 Shell' and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The main area of the window contains a series of Python commands and their outputs. The commands are: `a=5`, `b=3`, `print(a+b)`, `print(a-b)`, `print(a*b)`, `print(a/b)`, `print(a//b)`, `print(a%b)`, and `print(a**b)`. The outputs are: `8`, `2`, `15`, `1.6666666666666667`, `1`, `2`, and `125`. The window ends with three empty prompt lines and a cursor. The status bar at the bottom right shows 'Ln: 61 Col: 4'.

```
Python 3.5.3 Shell
File Edit Shell Debug Options Window Help
>>> a=5
>>> b=3
>>>
>>> print(a+b)
8
>>> print(a-b)
2
>>> print(a*b)
15
>>> print(a/b)
1.6666666666666667
>>> print(a//b)
1
>>> print(a%b)
2
>>> print(a**b)
125
>>>
>>>
>>> |
Ln: 61 Col: 4
```

산술 연산자 연습 2)

```
a=5
b=3

print("덧셈결과: %d + %d = %d" % (a, b, a+b))
print("뺄셈결과: %d - %d = %d" % (a, b, a-b))
print("곱셈결과: %d * %d = %d" % (a, b, a*b))
print("나눗셈결과: %d / %d = %f" % (a, b, a/b))
print("나누기몫: %d // %d = %d" % (a, b, a//b))
print("나머지값: %d %% %d = %d" % (a, b, a%b))
print("제곱결과: %d** %d = %d" % (a, b, a**b))
```

Ln: 12 Col: 0

산술 연산자 연습 3) – 2)에서 수정

```
a=int(input("첫번째 숫자:"))
b=int(input("두번째 숫자:"))

print("덧셈결과: %d + %d = %d" % (a, b, a+b))
print("뺄셈결과: %d - %d = %d" % (a, b, a-b))
print("곱셈결과: %d * %d = %d" % (a, b, a*b))
print("나눗셈결과: %d / %d = %f" % (a, b, a/b))
print("나누기몫: %d // %d = %d" % (a, b, a//b))
print("나머지값: %d %% %d = %d" % (a, b, a%b))
print("제곱결과: %d** %d = %d" % (a, b, a**b))
```

Ln: 12 Col: 0

대입 연산자

대입 연산자	사용 예	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

- ◆ **a는 10에서 시작하여 프로그램이 진행될수록 값이 누적됨**

```
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
```

대입 연산자 연습 1)

```
a = 10

a = a + 5
print("5 더하기 후 결과: %d" % a)

a = a - 5
print("5 빼기기 후 결과: %d" % a)

a = a * 5
print("5 곱하기 후 결과: %d" % a)

a = a / 5
print("5 나누기 후 결과: %f" % a)

a = a // 5
print("5 나눈 몫 대입 후 : %d" % a)

a = a % 5
print("5 나눈 나머지 대입 후: %d" % a)

a = a ** 5
print("5 제곱 대입 후: %d" % a)
```

Ln: 23 Col: 0

대입 연산자 연습 2) – 1)에서 수정

```
a = 10

a += 5
print("5 더하기 후 결과: %d" % a)

a -= 5
print("5 빼기기 후 결과: %d" % a)

a *= 5
print("5 곱하기 후 결과: %d" % a)

a /= 5
print("5 나누기 후 결과: %f" % a)

a //= 5
print("5 나눈 몫 대입 후 : %d" % a)

a %= 5
print("5 나눈 나머지 대입 후: %d" % a)

a **= 5
print("5 제곱 대입 후: %d" % a)
```

Ln: 23 Col: 0

연산자 사용

- ◆ 실수 연산을 수행하는 경우, 다른 형의 자료도 사용 가능
- ◆ 정수와 실수의 연산에서 정수는 실수로 변환되어 연산 수행

```
>>> 3 * 3.75 / 1.5
7.5
>>> 7.0 / 2
3.5
```

- ◆ 참고) IDLE과 같은 인터랙티브 모드에서는 최근에 출력된 결과물 재사용 가능. '_' 기호를 사용하면 이전의 결과물 재사용 (코드편집기에서는 사용 안됨)

```
>>> tax = 12.5/100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> _ - price
12.5625
```

문자열 연산

◆ 문자열도 숫자처럼 연산자 적용 가능

- 문자열 더하기(+)
- 문자열 곱하기 (*)

```
>>>
>>>
>>> ss1 = "hello "
>>> ss2 = "world"
>>>
>>> ss = ss1 + ss2
>>> ss
'hello world'
>>>
>>> ss = ss1*3
>>> ss
'hello hello hello '
>>>
>>> len(ss1)
6
>>>
```

◆ 문자열 길이 구하기 : len(문자열)

문자열과 숫자의 상호 변환

- ◆ 문자열을 숫자로 변환하기 위해서는
int(문자) 함수에 의해서 정수로,
float(문자) 함수에 의해서 실수로 변경

[illegible]

출력 결과

101 101.123 10000000000000000000000000

- ◆ 숫자를 문자열로 변환하기 위해서는 **str(숫자)** 함수를 사용

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

출력 결과

'1001'

```
'100.1231'
```

int(), float() 연습 1)

```
s1 = "100"    # 이건 문자열
s2 = "3.14"   # 이것도 문자열

result = int(s1)+1
print("정수 더하기: %d" % result)

result = float(s2)+1
print("실수 더하기: %f" % result)
print("실수 더하기: %.2f (소수점 두 자리)" % result)
```

Ln: 10 Col: 0

int(), float() 연습 2) – 1)에서 수정

```
s1 = input("첫번째 수를 입력하세요(정수): ") # 키보드 입력 (문자열)
s2 = input("두번째 수를 입력하세요(실수): ") # 키보드 입력 (문자열)

result = int(s1)+1
print("정수 더하기: %d" % result)

result = float(s2)+1
print("실수 더하기: %f" % result)
print("실수 더하기: %.2f (소수점 두 자리)" % result)
```

Ln: 10 Col: 0

int(), float() 연습 3) – 2)에서 수정

```
num1 = int(input("첫번째 수를 입력하세요(정수): "))    #숫자1
num2 = float(input("두번째 수를 입력하세요(실수): "))   # 숫자2

result = num1+1
print("정수 더하기: %d" % result)

result = num2+1
print("실수 더하기: %f" % result)
print("실수 더하기: %.2f (소수점 두 자리)" % result)
```

Ln: 8 Col: 0

관계 연산자(비교 연산자)

관계 연산자(비교 연산자)

- ◆ 어떤 것이 큰지, 작은지, 같은지를 **비교**하는 것.
결과는 참(True)이나 거짓(False)
- ◆ 주로 조건문(if)이나 반복문(for, while)에서 사용

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

- ◆ 관계연산자 종류

관계 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 예

```
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 비교하기 위한 관계 연산자 **==**를 사용 시,
=을 하나만 쓰는 경우 → 오류발생 (흔히 하는 실수!!!!)

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

관계 연산자 연습 1)

```
a = 100
b = 200

print("a 값: %d \t b 값: %d" % (a, b))

print("a와 b가 같습니까? ", a==b)
print("a와 b가 다른니까? ", a!=b)
print("a가 b보다 큼니까? ", a>b)
print("a가 b보다 작습니까? ", a<b)
print("a가 b보다 크거나 같나요? ", a>=b)
print("a가 b보다 작거나 같나요? ", a<=b)
```

Ln: 12 Col: 0

관계 연산자 연습 2) – 1)에서 수정

```
a = int(input("첫번째 수를 입력하세요:"))
b = int(input("두번째 수를 입력하세요:"))

print("a 값: %d \t b 값: %d" % (a, b))

print("a와 b가 같습니까? ", a==b)
print("a와 b가 다릅니까? ", a!=b)
print("a가 b보다 큼니까? ", a>b)
print("a가 b보다 작습니까? ", a<b)
print("a가 b보다 크거나 같나요? ", a>=b)
print("a가 b보다 작거나 같나요? ", a<=b)
```

Ln: 12 Col: 0

문자 비교 연습

```
s1 = input("첫번째 단어: ")
s2 = input("두번째 단어: ")

print("== 결과는?", s1 == s2)
print("!= 결과는?", s1 != s2)
print("> 결과는?", s1 > s2)
print("< 결과는?", s1 < s2)
print(">= 결과는?", s1 >= s2)
print("<= 결과는?", s1 <= s2)
```

Ln: 9 Col: 0

문자 비교 연습 결과

```
첫번째 단어: ABCD
두번째 단어: ABCD
== 결과는? True
!= 결과는? False
> 결과는? False
< 결과는? False
>= 결과는? True
<= 결과는? True
>>>
>>>
```

```
첫번째 단어: ABCD
두번째 단어: EFG
== 결과는? False
!= 결과는? True
> 결과는? False
< 결과는? True
>= 결과는? False
<= 결과는? True
>>> |
```

Ln: 386 Col: 4

ASCII Table

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL null	0x20	32	Space	0x40	64	@	0x60	96	~
0x01	1	SOH Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT End of transmission	0x24	36	\$	0x44	68	D	0x64	100	d
0x05	5	ENQ Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS Backspace	0x28	40	(0x48	72	H	0x68	104	h
0x09	9	TAB Horizontal tab	0x29	41)	0x49	73	I	0x69	105	i
0x0A	10	LF New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1 Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2 Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3 Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4 Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC Escape	0x3B	59	;	0x5B	91	[0x7B	123	{
0x1C	28	FS File separator	0x3C	60	<	0x5C	92	\	0x7C	124	
0x1D	29	GS Group separator	0x3D	61	=	0x5D	93]	0x7D	125	}
0x1E	30	RS Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

is 연산자

- ♦ 값이 같은지 비교하는 '==' 연산자와 'is' 연산자
- ♦ is 연산자 : 두 개의 주소값이 같은지 비교
- ♦ is 연산이 좀 더 빠름

```
>>>
>>> a = 5
>>> b = 5
>>>
>>> a == b
True
>>> a is b
True
>>>
>>>
>>> a = [1,2,3,4,5]
>>> b = [1,2,3,4,5]
>>>
>>> a == b
True
>>> a is b
False
>>>
```

주소 위치값 확인

- ◆ id (변수명)

```
>>>
>>> a = [1,2,3,4,5]
>>> b = [1,2,3,4,5]
>>>
>>> a == b
True
>>> a is b
False
>>>
>>>
>>>
>>>
>>> print ("a 저장 공간 주소 위치 : ", id(a))
a 저장 공간 주소 위치 : 57448848
>>> print ("b 저장 공간 주소 위치 : ", id(b))
b 저장 공간 주소 위치 : 63839008
>>>
>>>
```

- ◆ True, False, None과 같이 자주 쓰이는 값들을 Python에서는 같은 주소값에 위치 시킴

주소 위치값 확인

◆ '변수 = 변수'의 주소값 확인

```
>>>
>>> a = [1,2,3,4,5]
>>> b = a
>>>
>>> a == b
True
>>> a is b
True
>>>
>>> print ("a 저장 공간 주소 위치 : ", id(a))
a 저장 공간 주소 위치 : 13910704
>>> print ("b 저장 공간 주소 위치 : ", id(b))
b 저장 공간 주소 위치 : 13910704
>>>
>>>
>>> a = 5
>>> b = 5
>>>
>>> print ("a 저장 공간 주소 위치 : ", id(a))
a 저장 공간 주소 위치 : 1734793456
>>> print ("b 저장 공간 주소 위치 : ", id(b))
b 저장 공간 주소 위치 : 1734793456
>>>
>>>
```

논리 연산자

논리 연산자

- 참과 거짓(논리값)을 이용한 연산기능으로, 여러 조건들을 검사하고 증명

논리 연산자	의미	설명	사용 예
and	~이고, 그리고(AND)	둘 다 참이어야 참	$(a > 100) \text{ and } (a < 200)$
or	~이거나, 또는(OR)	둘 중 하나만 참이어도 참	$(a == 100) \text{ or } (a == 200)$
not	~아니다, 부정(NOT)	참이면 거짓, 거짓이면 참	$\text{not}(a < 100)$

- 진리표

입력값		A 그리고 B	A 또는 B	A가 아니다
A	B	(A and B)	(A or B)	(!A)
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

논리 연산자 예

```
>>> True and False
False
>>> True or False
True
>>> not True
False
>>> not False
True
```

```
>>> a = 99
>>>
>>> (a<100) and (a<200)
True
>>> (a<90) and (a<100)
False
>>> (a>90) and (a<100)
True
>>> 90<a<100
True
```

```
>>> a = 99
>>>
>>> (a<90) or (a<100)
True
>>>
>>> not(a==100) and (a<200)
True
>>> not(a==99) and (a<200)
False
```

논리연산자 + 비교연산자

- ◆ 비교 연산자(is, is not, ==, !=, <, >, <=, >=) 먼저 판단
- ◆ 이 후 논리 연산자(not, and or) 판단 (연산자 우선순위 참고)

```
>>>  
>>> 10==10 and 10!=5  
True  
>>> 10 > 5 and 10 < 3  
False  
>>> not 10 > 5  
False  
>>> not 1 is 1.0  
True  
>>>
```


비트 연산자

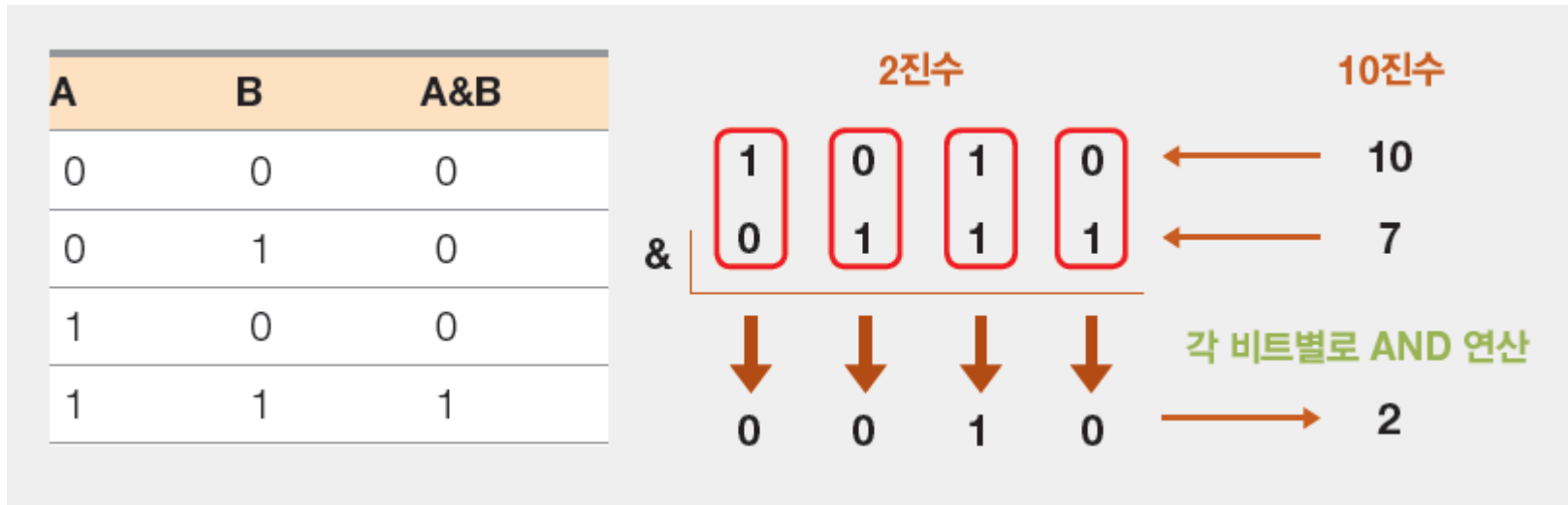
비트 연산자

- 주어진 수를 각 비트별 처리하는 연산자 (2진수 각 자리에 대한 연산 수행)

비트 연산자	설명	의미
&	비트 논리곱(AND)	둘 다 1이면 1
	비트 논리합(OR)	둘 중 하나만 1이면 1
^	비트 배타적 논리합(XOR)	둘이 같으면 0, 다르면 1
~	비트 부정	1은 0으로, 0은 1로 변경
<<	비트 이동(왼쪽)	비트를 왼쪽으로 시프트(Shift)함
>>	비트 이동(오른쪽)	비트를 오른쪽으로 시프트(Shift)함

비트 논리곱(&) 연산자

◆ 비트 논리곱 예 : 10 & 7



◆ 10 & 7의 결과는 2

- ◆ 10진수를 2진수로 변환한 다음 각 비트마다 AND 연산을 수행.
그 결과, 2진수 0010이 되고 10진수로는 2가 됨

비트 논리곱(&) 예

10 & 7

123 & 456

0xFFFF & 0x0000

출력 결과

2 72 0

- ◆ 10진수 123 & 456
- ◆ 2진수 111 1011과
1 1100 1000의 비트 논리곱(&
결과: 100 1000
- ◆ 10진수로 72
- ◆ 16진수 0xFFFF & 0x0000
- ◆ 2진수 1111 1111 1111 1111와
0000 0000 0000 0000의
비트 논리곱(&
결과: 0000 0000 0000 0000
- ◆ 0이 출력
- ◆ 0과 비트 논리곱을 수행하면 어떤 수든 무조건 0

비트 논리합(|) 연산자

- ◆ 비트 논리합 예 : 10 | 7



- ◆ 각 비트의 논리합 결과는 2진수 1111이며, 이는 10진수 15가 됨

비트 논리합(|) 예

10 | 7

123 | 456

0xFFFF | 0x0000

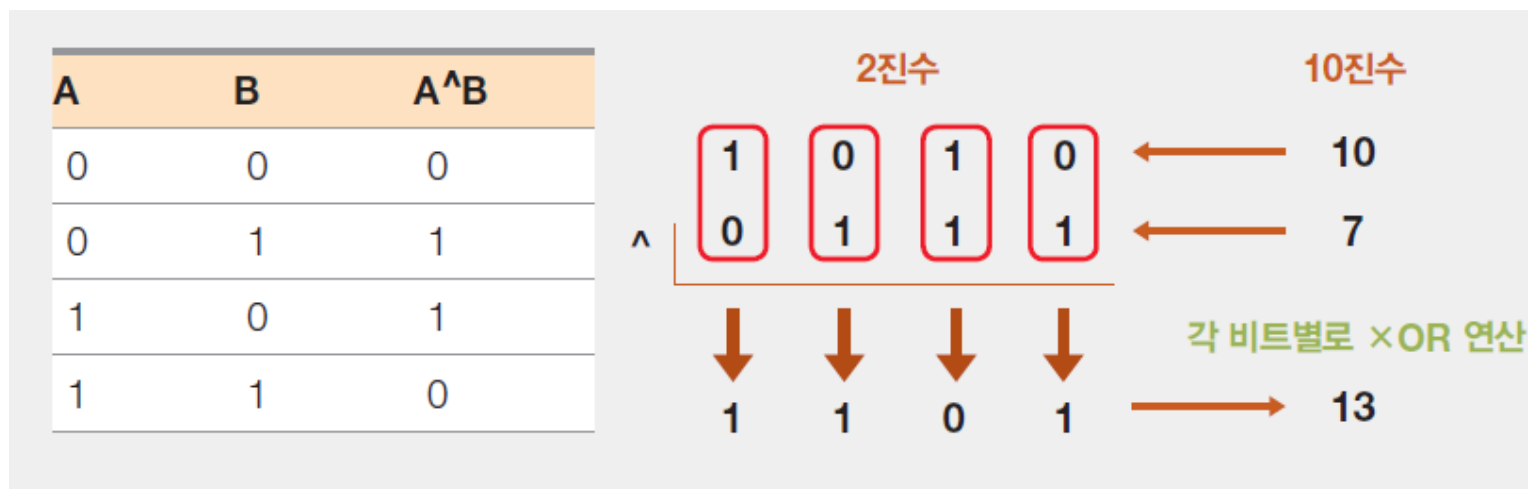
출력 결과

15 507 65535

- ◆ 0xFFFF와 0x0000의 비트 논리합은 0xFFFF 임.
그러므로 16진수 0xFFFF → 10진수 65,535가 됨

비트 배타적 논리합(^) 연산자

- ◆ Exclusive OR : 두 값이 같으면 0, 다르면 1
- ◆ 1^1 , $0^0 \rightarrow \text{False}(0)$
 1^0 , $0^1 \rightarrow \text{True}(1)$



- ◆ 10과 7의 각 비트 배타적 논리합 결과는 1101 \rightarrow 10진수 13

비트 배타적 논리합(^) 예

$10 \wedge 7$

$123 \wedge 456$

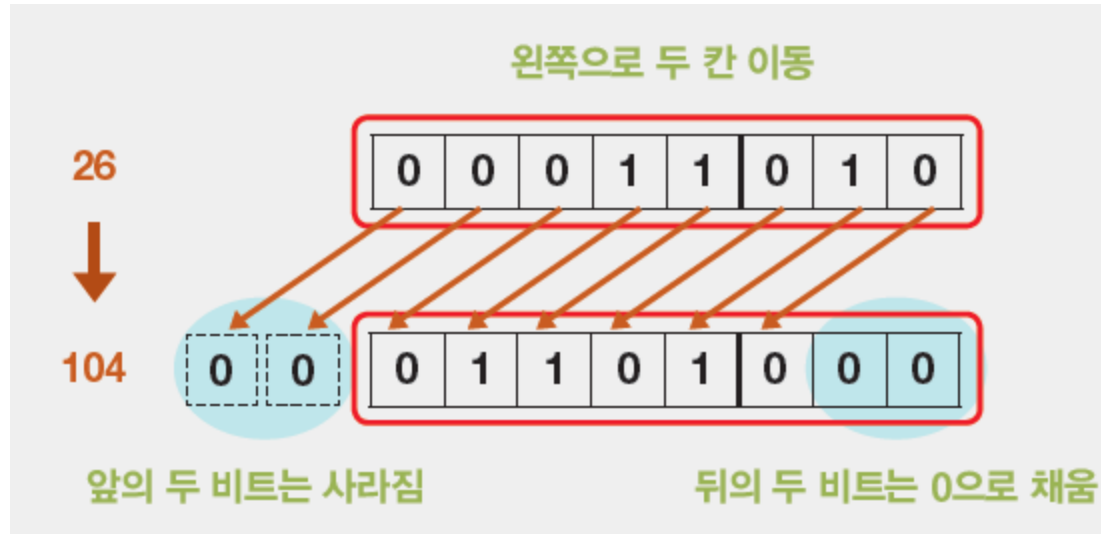
$0xFFFF \wedge 0x0000$

출력 결과

13 435 65535

왼쪽 시프트(<<) 연산자

- ◆ 나열된 비트를 왼쪽으로 시프트(Shift)해주는 연산자
- ◆ 26을 왼쪽으로 두 칸 시프트



- ◆ 앞의 두 00은 떨어져나가고, 뒤에 빈 두 칸에는 00으로 채움.
결과는 26에서 104로 바뀌었는데,
이는 **왼쪽으로 시프트 할 때마다 2^n 을 곱한 효과**가 나기 때문($26 \times 2^2 = 104$)
- ◆ 즉, 왼쪽으로 1회 시프트 할 때는 2^1 을, 2회에는 2^2 을, 3회에는 2^3 을...
곱한 것과 같음

왼쪽 시프트(<<) 연산자 예

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

출력 결과

20 40 80 160

- ◆ **시프트 할 때마다**

$$10 \times 2^1 = 20,$$

$$10 \times 2^2 = 40,$$

$$10 \times 2^3 = 80,$$

$$10 \times 2^4 = 160 \text{ 의 결과가 나옴}$$

오른쪽 시프트(>>) 연산자

- ◆ 나열된 비트를 오른쪽으로 시프트(Shift)해주는 연산자
- ◆ 26을 오른쪽으로 두 칸 시프트



- ◆ 오른쪽의 두 비트는 떨어져나가고, 왼쪽의 두 비트에는 부호 비트(양수는 0이, 음수는 1)가 채워짐. 이는 2^n 으로 나눈 효과
- ◆ 또한, 시프트 연산은 정수만 연산하므로 몫만 남음($26 / 2^2 = 6$). 즉, 오른쪽으로 1회 시프트 할 때는 2^1 , 2회에는 2^2 , 3회에는 2^3 으로... 나누는 효과가 남

오른쪽 시프트(>>) 연산자 예

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

출력 결과

```
5 2 1 0
```

- ◆ 시프트 할 때마다

$$10/2^1=5,$$

$$10/2^2=2,$$

$$10/2^3=1,$$

$$10/2^4=0 \text{ 의 결과가 나옴}$$

비트 (~) 연산자

- ◆ 두 수에 대해 연산하는 것이 아니라, **각 비트를 반대로** 만드는 연산자
- ◆ 즉, 각 비트에 대해 0은 1로, 1은 0으로 바꿈.
예) $\sim 0000 \rightarrow 1111$, $\sim 0101 \rightarrow 1010$

- ◆ 이렇게 반전된 값을 '1의 보수'라 함.

- ◆ 비트 부정 연산자는 어떤 값의 반대 부호의 값을 찾고자 할 때 활용

```
a = 12345
```

```
 $\sim a + 1$ 
```

출력 결과

```
-12345
```

- ◆ why +1을 해주나??

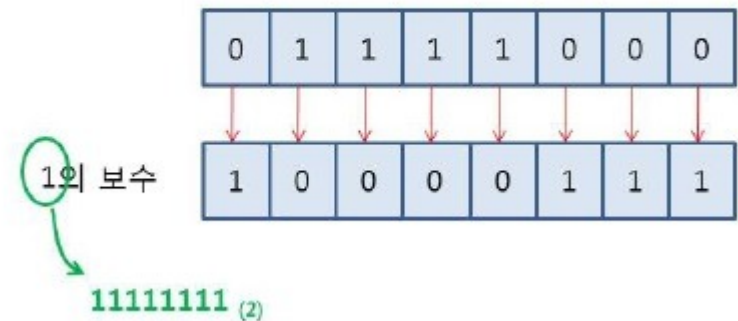
보수의 개념

- ◆ 컴퓨터에서 보수는 음수를 표현하는 데 사용
- ◆ 보수의 의미 : 상호 보완하는 수로, 임의의 수를 보완해주는 다른 임의의 수

1의 보수 계산법

◆ 1의 보수

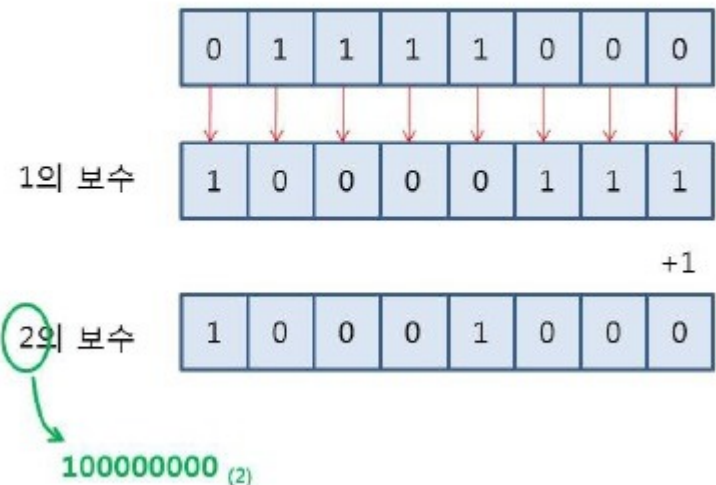
- $0 \rightarrow 1, 1 \rightarrow 0$ 으로 변환
 $0111\ 1000 \rightarrow 1\text{의 보수} = 1000\ 0111$



2의 보수 계산법

◆ 2의 보수

- $1\text{의 보수} + 1 = 2\text{의 보수}$
 $0111\ 1000 \rightarrow 2\text{의 보수}$
 $= 1\text{의 보수} + 1$
 $= 1000\ 0111 + 1$
 $= 1000\ 1000$



2진 음수를 표시하는 방법 4가지 1)

- 1) 부호와 크기(sign and magnitude) : 부호와 크기 방법은 양의 정수를 그대로 사용하고, 최상위 부호비트에 1을 사용

음의 정수 표현: 부호와 크기(sign and magnitude)

$$120_{10} = 1111000_2$$

부호 비트	값의 범위
0	$0 \sim (2^{n-1}-1)$
1	$-(2^{n-1}-1) \sim -0$



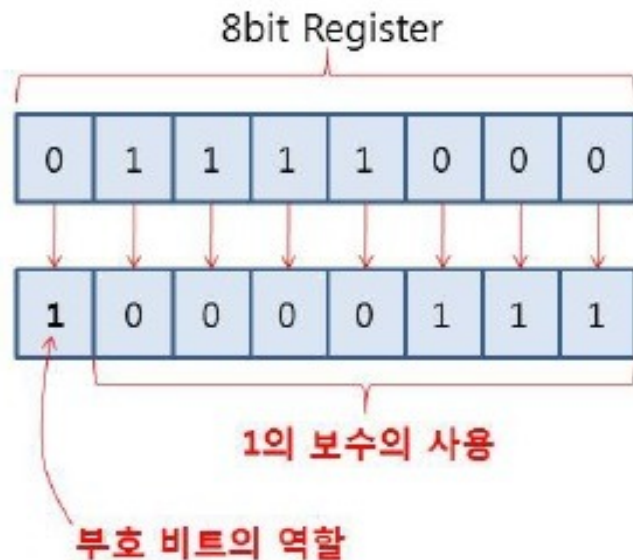
2진 음수를 표시하는 방법 4가지 2)

- 2) 1의 보수(1's complement) : 부호 비트 1과 2진수의 절댓값에 대한 1의 보수를 사용

음의 정수 표현: 1의 보수(1's complement)

$$120_{10} = 1111000_2$$

부호 비트	값의 범위
0	$0 \sim (2^{n-1}-1)$
1	$-(2^{n-1}-1) \sim -0$



2진 음수를 표시하는 방법 4가지 3)

- 3) 2의 보수(2's complement) : 1의 보수에 +1
(+0, -0을 방지, 일반적인 컴퓨터에서 사용하는 방식)

음의 정수 표현: 2의 보수(2's complement)

$$120_{10} = 1111000_2$$

부호 비트	값의 범위
0	$0 \sim (2^{n-1}-1)$
1	$-(2^{n-1}) \sim -1$



2진 음수를 표시하는 방법 4가지 4)

- 4) Exceed n 방식 (비균형 방식) : 중간값(0111111)을 0으로 사용하여 그 위로는 음수 아래로는 양수

음의 정수 표현: 비균형(exceed n)

이진 표현	양수	Exceed 127
0000 0000	0	-127
0000 0001	1	-126
0000 0010	2	-125
...
0111 1111	127	0
1000 0000	128	1
...
1111 1111	255	128

정수 표현 방법 비교

Binary	Unsigned	Sign and Mag.	1's comp.	2's comp.	Exceed 127
0000 0000	0	0	0	0	-127
0000 0001	1	1	1	1	-126
0000 0010	2	2	2	2	-125
...
0111 1101	125	125	125	125	-2
0111 1110	126	126	126	126	-1
0111 1111	127	127	127	127	0
1000 0000	128	-0	-127	-128	1
1000 0001	129	-1	-126	-127	2
1000 0010	130	-2	-125	-126	3
...
1111 1101	253	-125	-2	-3	126
1111 1110	254	-126	-1	-2	127
1111 1111	255	-127	-0	-1	128

네 가지 정수 표현 방법 비교

표현 방법	값의 범위	0(zero) 표현	표현 가능 개수
부호 비트	$-2^{n-1}+1 \sim 2^{n-1}-1$	+0, -0	2^{n-1}
1의 보수	$-2^{n-1}+1 \sim 2^{n-1}-1$	+0, -0	2^{n-1}
2의 보수	$-2^{n-1} \sim 2^{n-1}-1$	0	2^n
Exceed $2^{n-1}-1$	$-2^{n-1}+1 \sim 2^{n-1}$	0	2^n

파이썬에서 (~)연산 결과

```
>>> a = 67
>>> bin(a)
'0b1000011'
>>> bin(~a)
'-0b1000100'
>>> int(~a)
-68
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>> a = 7
>>> bin(a)
'0b111'
>>> bin(~a)
'-0b1000'
>>> int(~a)
-8
```

연산자 우선순위

우선순위	연산자	설명
1	() [] {}	괄호, 리스트, 딕셔너리, 세트 등
2	**	지수
3	+ - ~	단항 연산자
4	* / % //	산술 연산자
5	+ -	산술 연산자
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	논리 연산자
15	or	논리 연산자
16	if ~ else	비교식

단항 연산자

◆ 피연산자의 개수에 따른 분류

- 단항 연산자: +(부호), -(부호) (C언어에서... ++, -- 즉, 증감연산자도 단항 연산자임)
- 이항 연산자: 피연산자가 2개인 산술연산자 (+, -, *, /, ...등등)
- 삼항 연산자
 - C언어의 경우: condition? true: false;
 - Python (2.5버전 이상)의 경우: true if condition else false

- ◆ '2 + 3 * -4' 라는 수식인 경우...
= (2 + (3 * (-4))) 와 같다. 즉, 연산 결과는 -10
- ◆ 결합 순서를 변경하기 위해서는 괄호 사용
- ◆ '(2 + 3) * -4' 수식의 결과는 -20

if문(조건문)

if문 사용법

```
1 a=200
2
3 if a<100 :
4     ➡ print("100보다 작군요")
5     print("거짓이므로 이 문장은 안 보이겠죠?")
6
7     print("프로그램 끝")
```

거짓이므로 이 문장은 안 보이겠죠?
프로그램 끝

- ◆ 들여쓰기(탭)를 하지 않아, 실행 하지 않아야 할 5행까지 실행 됨.
→ 줄 바꿈을 수정하여 실행

if문 사용법

```
1 a=200
2
3 if a<100 :
4     print("100보다 작군요")
5     ➡ print("거짓이므로 이 문장은 안보이겠죠?")
6
7 print("프로그램 끝")
```

프로그램 끝

- ◆ 범할 수 있는 오류 : Tab key가 아닌 Space key 사용

```
if a<100 :
    print("100보다 작군요")
    print("거짓이므로 이 문장은 안보이겠죠?")
```

if ~ else 문 사용법

```
1  a=200
2
3  if a<100 :
4      print("100보다 작군요.")
5      print("참이면 이 문장도 보이겠죠?")
6  else :
7      print("100보다 크군요.")
8      print("거짓이면 이 문장도 보이겠죠?")
9
10 print("프로그램 끝!")
```

100보다 크군요.
거짓이면 이 문장도 보이겠죠?
프로그램 끝!

중첩 if문 예시 1)

```
1 a=75
```

```
2
```

```
3 if a>50 :
```

```
4     if a<100 :
```

```
5         print("50보다 크고 100보다 작군요.")
```

```
6     else :
```

```
7         print("와~~ 100보다 크군요.")
```

```
8 else :
```

```
9     print("에고~ 50보다 작군요..")
```

→ if 참인 경우 ($a>50$)

→ if 거짓인 경우 ($a\leq 50$)

50보다 크고 100보다 작군요.

- ◆ 3행에서 $a > 50$ 이면 참이므로
들여쓰기가 된 부분(4행~7행)의 내용을 실행
그 안에서 a 가 100보다 작으므로 5행을 출력

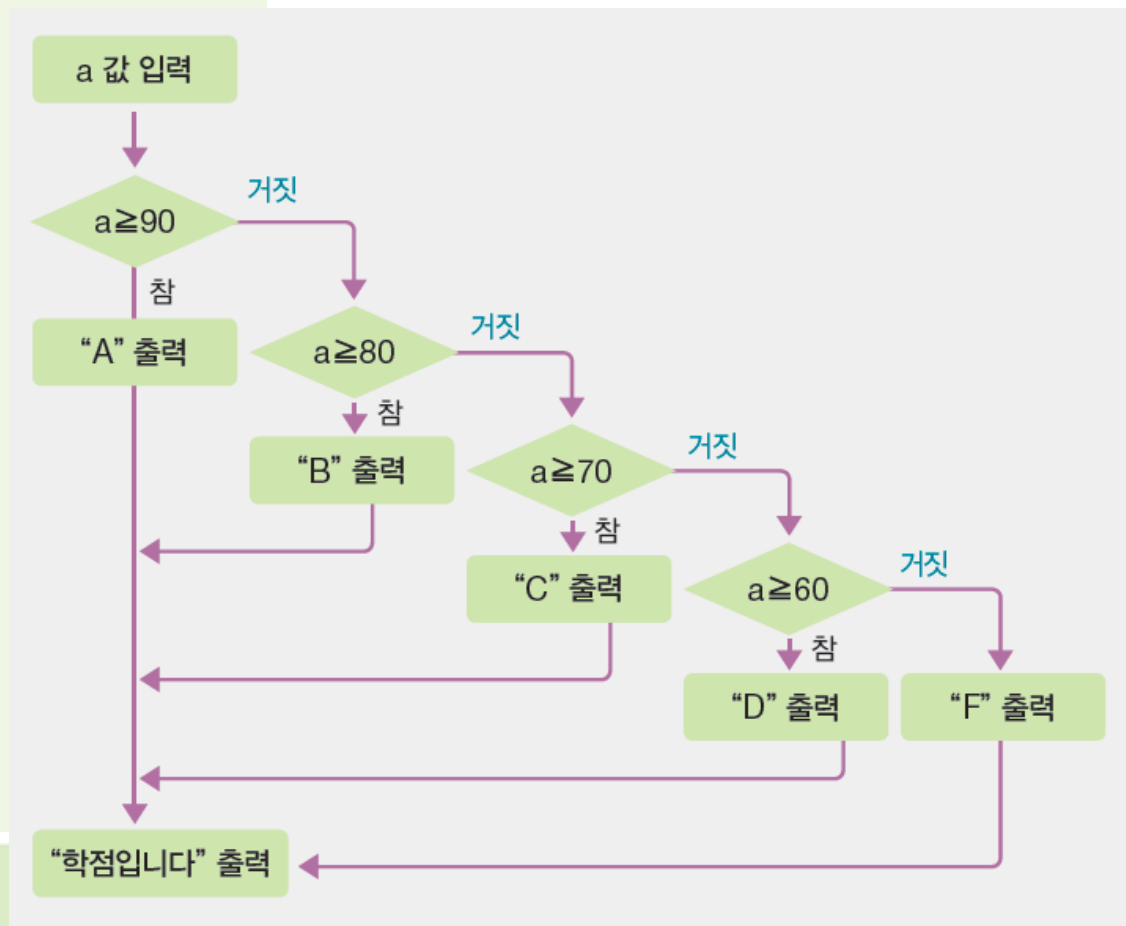
중첩 if문 예시 2)

```
1 score=int(input("점수를 입력하세요 : "))
2
3 if score>=90 :
4     print("A")
5 else :
6     if score>=80 :
7         print("B")
8     else :
9         if score>=70 :
10            print("C")
11        else :
12            if score>=60 :
13                print("D")
14            else :
15                print("F")
16
17 print("학점입니다. ^^")
```

점수를 입력하세요 : 85

B

학점입니다. ^^



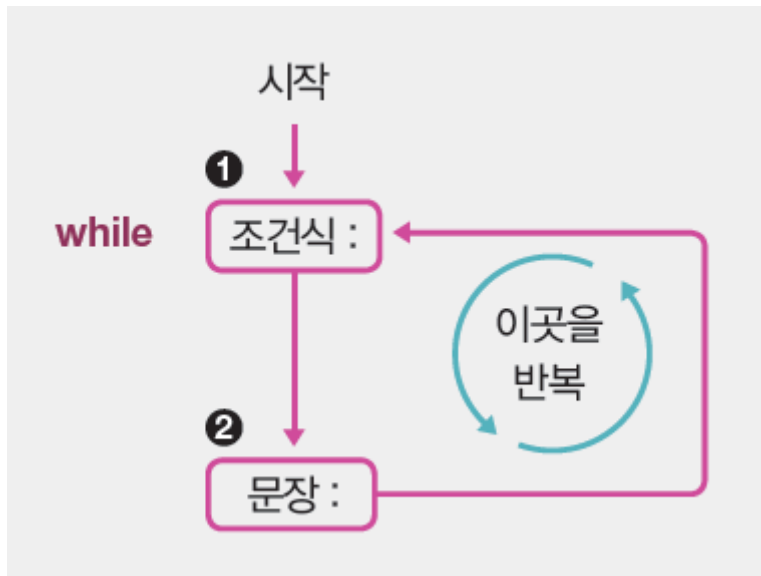
if ~ elif ~ else 문

```
1 score=int(input("점수를 입력하세요 : "))
2
3 if score>=90 :
4     print("A")
5 elif score>=80 :
6     print("B")
7 elif score>=70 :
8     print("C")
9 elif score>=60 :
10    print("D")
11 else :
12    print("F")
13
14 print("학점입니다. ^^")
```

while문 (반복문)

while문

- ♦ while문 안의 조건식을 확인하여 값이 참이면 '문장'을 수행. 조건식이 참인 동안 계속 반복



형식:

변수 = 시작값

while 변수값 < 끝값 :

이 부분을 반복

변수 = 변수 + 증가값

while문 활용 합계 구하기

◆ 1에서 10까지의 합

```
i, sum = 0, 0  
  
i=1  
while i<11 :  
    sum = sum + i  
    i += 1  
  
print("1에서 10까지의 합 : %d" % sum)
```

1에서 10까지의 합 : 55

while문 활용 (무한 루프)

- ◆ 입력한 두 수의 합계를 계산하는 프로그램 (끝내려면 'Ctrl+C')

```
sum = 0
a, b = 0, 0

while True :
    a = int(input("첫 번째 수 입력 : "))
    b = int(input("두 번째 수 입력 : "))
    sum = a+b
    print("%d+%d=%d" % (a, b, sum))
```

```
첫 번째 수 입력 : 55
두 번째 수 입력 : 22
55+22=77
첫 번째 수 입력 : 77
두 번째 수 입력 : 128
77+128=205
첫 번째 수 입력 :
```

for문 (순회문)

for문 사용

```
for i in range(0, 3, 1) :  
    print("안녕하세요? for문을 공부중입니다. ^^")
```

출력 결과

```
안녕하세요? for문을 공부중입니다. ^^  
안녕하세요? for문을 공부중입니다. ^^  
안녕하세요? for문을 공부중입니다. ^^
```

◆ for i in range(0, 3, 1) :

형식:

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

for 문의 동작

◆ for i in range(0, 3, 1) :

- range(0, 3, 1) : 지정된 범위의 값을 반환 → 0부터 2까지 1씩 증가
- 순회할 자료구조

◆ range(0, 3, 1) = [0, 1, 2]

```
for i in [0, 1, 2] :  
    print("안녕하세요? for문을 공부중입니다. ^^")
```

◆ 변수 i에 0,1,2를 차례로 대입하며 3회 반복

```
1회 : i에 0을 대입한 후 print() 수행  
2회 : i에 1을 대입한 후 print() 수행  
3회 : i에 2를 대입한 후 print() 수행
```

for 문의 사용

- ◆ range() 함수에서...
i값의 시작 값을 2로 하고, 0이 될 때까지, 1씩 줄여가며...
print() 함수를 세 번 실행하는 프로그램

```
for i in range(2, -1, -1) :  
    print("%d : 안녕하세요? for문을 공부중입니다. ^^" % i )
```

출력 결과

```
2 : 안녕하세요? for문을 공부중입니다. ^^  
1 : 안녕하세요? for문을 공부중입니다. ^^  
0 : 안녕하세요? for문을 공부중입니다. ^^
```

- ◆ i 값 : 2 → 1 → 0
하나 더 진행한 '-1'로 설정해야 '0'까지 처리

for 문의 사용

- ◆ 1~5까지 숫자들을 차례대로 출력

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

출력 결과

1 2 3 4 5

- ◆ 출력 결과가 한 줄에 나온 이유는...
print() 함수의 마지막에 **end=" "**를 썼기 때문
- ◆ 파이썬 3 버전 이상의 경우 줄바꿈 문자를 제거하기 위해서
위와 같이 **끝 문자를 지정**할 수 있는 end 파라미터를 설정하면 됨
- ◆ 지정하지 않으면 디폴트로 \n(new line) 문자가 셋팅

for 문 활용 합계 구하기

◆ 1부터 10까지의 합 구하기

```
i, sum = 0, 0  
  
for i in range(1, 11, 1) :  
    sum = sum + i  
  
print("1에서 10까지의 합 : %d" % sum)
```

1에서 10까지의 합 : 55

for 문 활용 합계 구하기

◆ 500과 1000 사이에 있는 홀수의 합 구하기

- $501+503+505+507+509+601+\dots+999 = ?$

1. for문 변수 `i`, 합계 변수 `sum` 선언 및 초기화
2. `i` 값 변화: 501(시작값, 홀수)부터 1000+1(끝값)까지 2씩 증가(홀수)
 - ① `sum` 값에 `i` 값(501~1000 사이 홀수) 더해줌
3. 합계 출력 (`print sum`)

```
i, sum = 0, 0

for i in range(501, 1001, 2) :
    sum = sum + i

print("500에서 1000까지 홀수의 합 : %d" % sum)
```

```
500에서 1000까지 홀수의 합 : 187500
```


for 문 활용 합계 구하기

◆ 입력한 값까지 for문으로 합계 구하기

- 사용자가 원하는 값을 입력하여 1부터 입력한 수까지의 합을 구하는 프로그램
- 끝값 : 입력한 수 +1

```
i, sum = 0, 0
num = 0

num = int(input("값 입력 : "))

for i in range(1, num+1, 1) :
    sum = sum + i

print("1에서 %d까지의 합 : %d" % (num, sum))
```

```
값 입력 : 100
1에서 100까지의 합 : 5050
```

입력한 숫자 단의 구구단 출력

```
몇 단 : 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
```

```
## loop 변수, 구구단 단수
i, dan = 0, 0

## 단수 입력
dan = int(input("몇 단 : "))

## 구구단 계산 및 출력
for i in range(1, 10, 1):
    print("%d x %d = %2d" % (dan, i, dan * i))
```

중첩 for 문 기본 코드

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

파이썬 출력... (i값 : 0, k값 : 0)

파이썬 출력... (i값 : 0, k값 : 1)

파이썬 출력... (i값 : 1, k값 : 0)

파이썬 출력... (i값 : 1, k값 : 1)

파이썬 출력... (i값 : 2, k값 : 0)

파이썬 출력... (i값 : 2, k값 : 1)

① i=0

② i=1

③ i=2

①-1 k=0

①-2 k=1

②-1 k=0

②-2 k=1

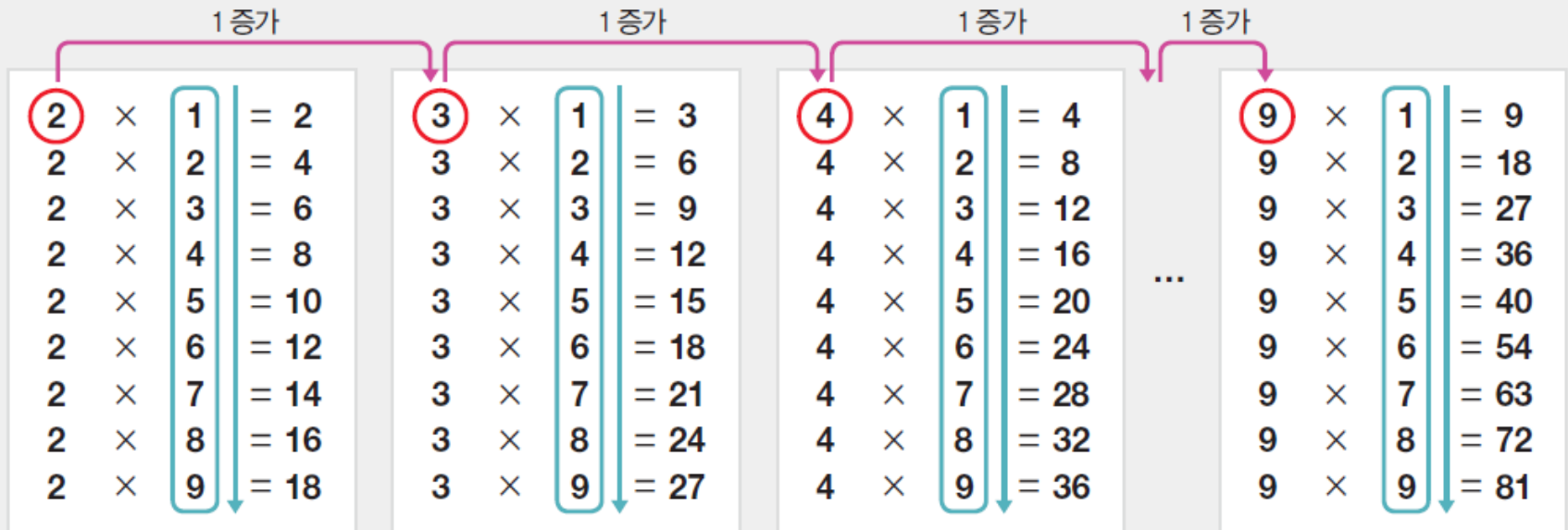
③-1 k=0

③-2 k=1

중첩 for문 활용

- ◆ 구구단 2단부터 9단까지 출력
 - 2~9단 (바깥 for문 : 변수 i)
 - 1~9 곱하는 수 (안쪽 for문 : 변수 k)

2~9까지 증가 후 종료(바깥 for문 : 변수 i)



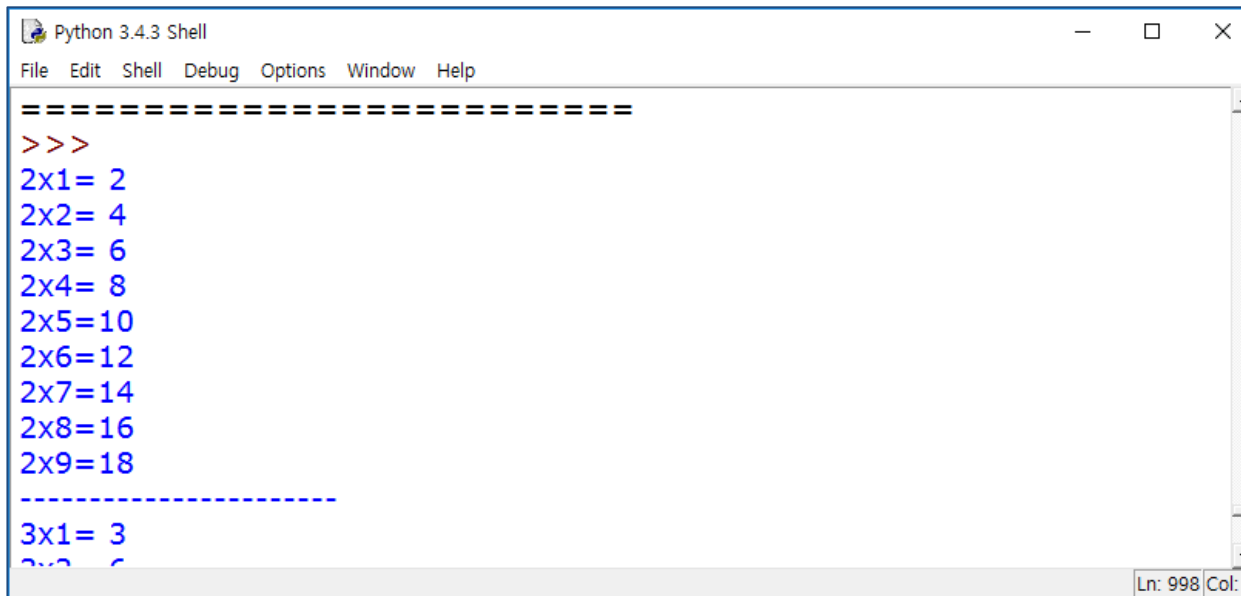
1~9까지 계속 반복해서 증가(안쪽 for문 : 변수 K)

중첩 for문을 활용한 구구단 출력

◆ 구구단 2단부터 9단까지 출력

```
i, k = 0, 0
```

```
for i in range(2, 10, 1) :  
    for k in range(1, 10, 1) :  
        print("%dx%d=%2d" % (i, k, i*k))  
    print("-----")
```



The screenshot shows a Python 3.4.3 Shell window with the following output:

```
=====
```

```
>>>  
2x1= 2  
2x2= 4  
2x3= 6  
2x4= 8  
2x5=10  
2x6=12  
2x7=14  
2x8=16  
2x9=18  
-----  
3x1= 3  
3x2= 6
```

The status bar at the bottom right indicates "Ln: 998 Col: 4".

break문

- ◆ 1~100까지 더하되, 누적 합계(sum)가 1000 이상이 되는 시작 지점을 구하는 프로그램

```
i, sum = 0, 0  
  
for i in range (1, 101) :  
    sum += i  
  
    if sum >= 1000 : # 합이 1000이 넘으면 for문 중지  
        break  
  
print("최초로 1000이 넘는 위치 : 1~ %d => 합 : %d" % (i, sum))
```

최초로 1000이 넘는 위치 : 1~ 45 => 합 : 1035

- ◆ range 함수에서...

range (1, 999) : 증감 숫자가 '1'인 경우 생략 가능

= range (1, 999, 1)

range (999) : 시작 숫자가 '0'인 경우도 생략 가능

= range (0, 999, 1)

continue문

- ◆ 1~100까지의 합을 구하되 3의 배수를 건너뛰고(=제외하고) 더하는 프로그램 (1 +2 +4 +5 +7 +8 +10 +... + 98 +100 = ???)

```
i, sum = 0, 0

for i in range (1, 101) :
    if i % 3 == 0 :
        continue # 3의 배수이면 sum 을 더하지 않고, 다음 for문 수행 (i+1)

    sum += i
    print("%d " % i, end="")

print("\n1~100의 합계(3배수 제외) : %d" % sum)
```

```
1 2 4 5 7 8 10 11 13 14 16 17 19 20 22 23 25 26 28 29 31 32 34 35 37 38 40
41 43 44 46 47 49 50 52 53 55 56 58 59 61 62 64 65 67 68 70 71 73 74 76 77
79 80 82 83 85 86 88 89 91 92 94 95 97 98 100
1~100의 합계(3배수 제외) : 3367
```

Any Questions...
Just Ask!

