# Basic Math

국민대학교 소프트웨어학부

# 수업목표

- Four basic operations
- 대입연산자
- Order of operations
- 비트 연산자
- More operators
- Really big and really small
- 진법변환
- 심화학습

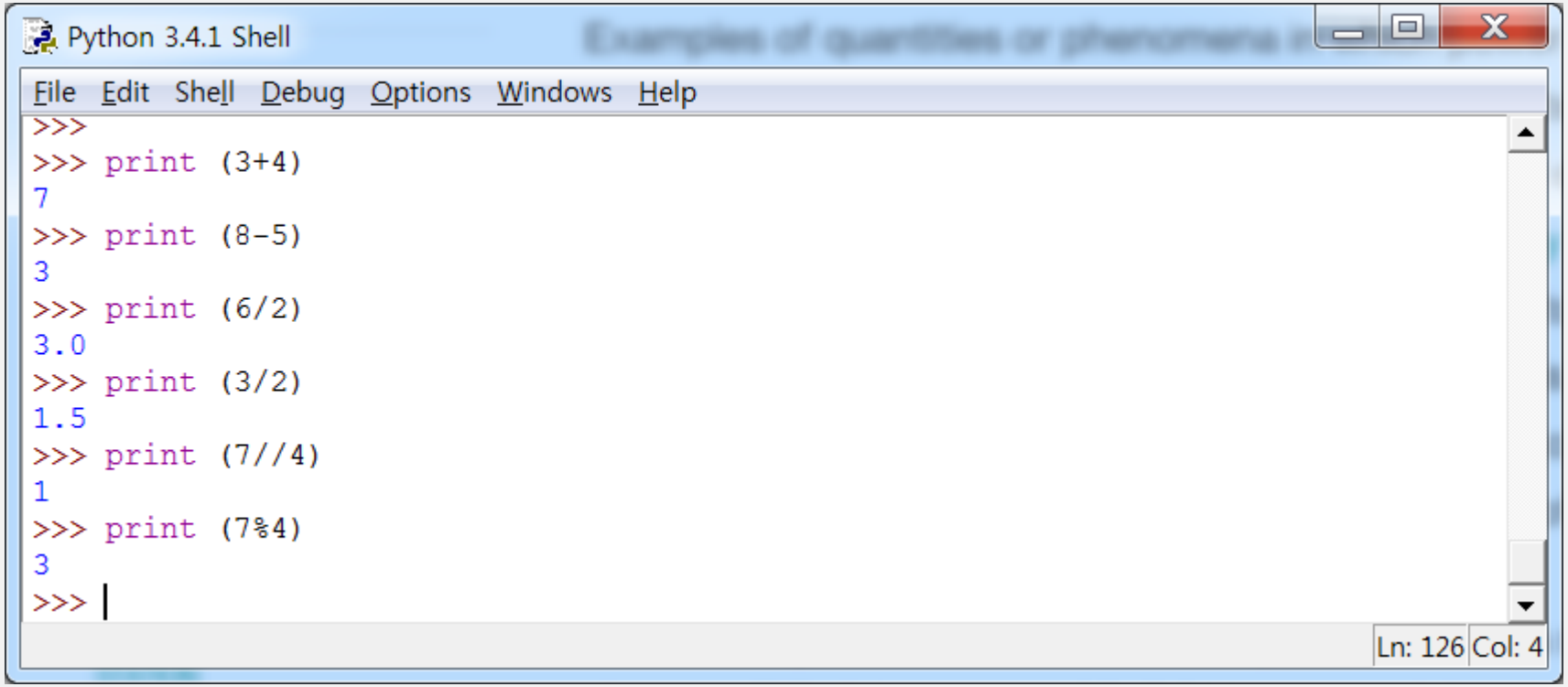
# Four basic operations

- 숫자
  - 파이썬은 크게 정수(int), 실수(float) 그리고 복소수(complex) 형태의 숫자를 표현할 수 있다.

| 자료형 | 설명 | 예 |
|---|---|---|
| int | 정수를 표현하는 자료형 | 123 |
| float | 실수를 표현하는 자료형 | 1.43 |
| complex | 복소수를 표현하는 자료형 | 5+4i |

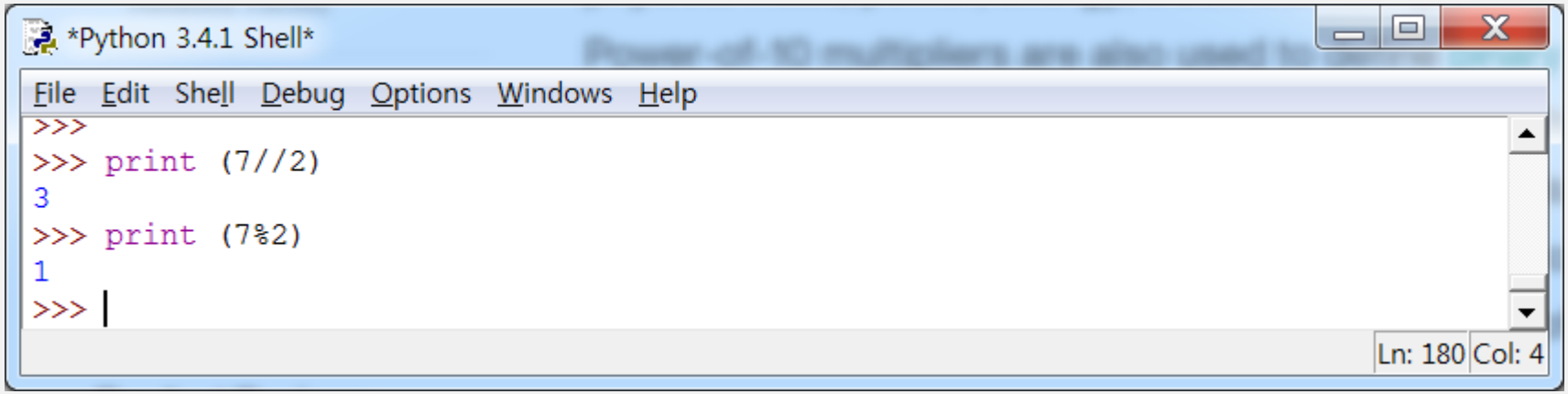
# Four basic operations

- 사칙연산(operations)과 연산자(operator)
  - 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/)



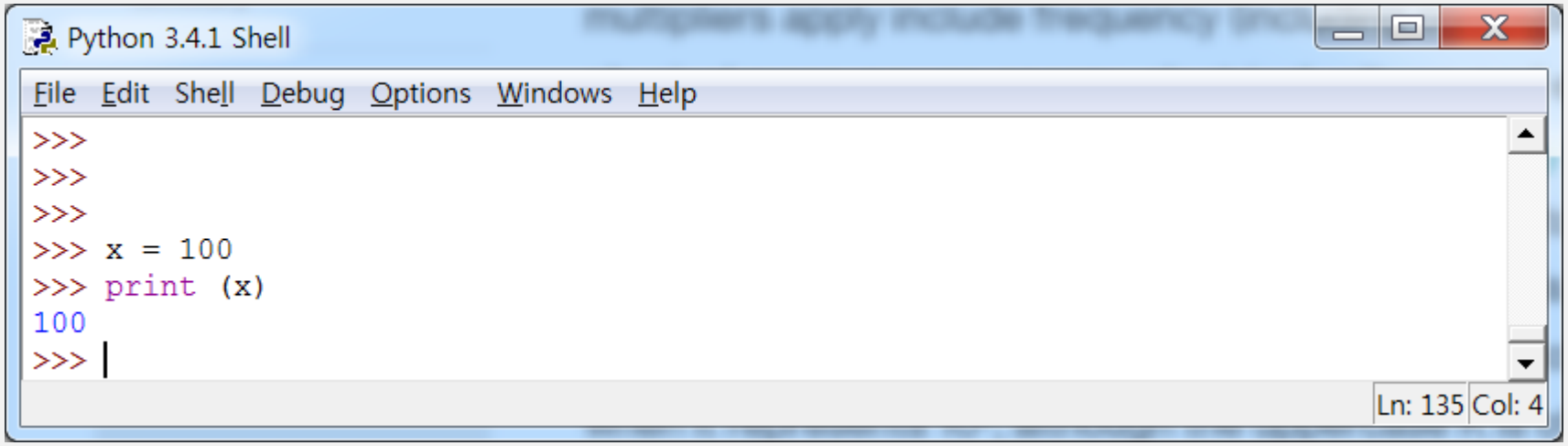
# Four basic operations

- 나누기 연산
  - 몫: //
  - 나머지(Modular): %



# 대입 연산자

- '='는 할당 연산자(assignment operator)
- 동일함을 나타내는 '=='과 구분할 줄 알아야 함

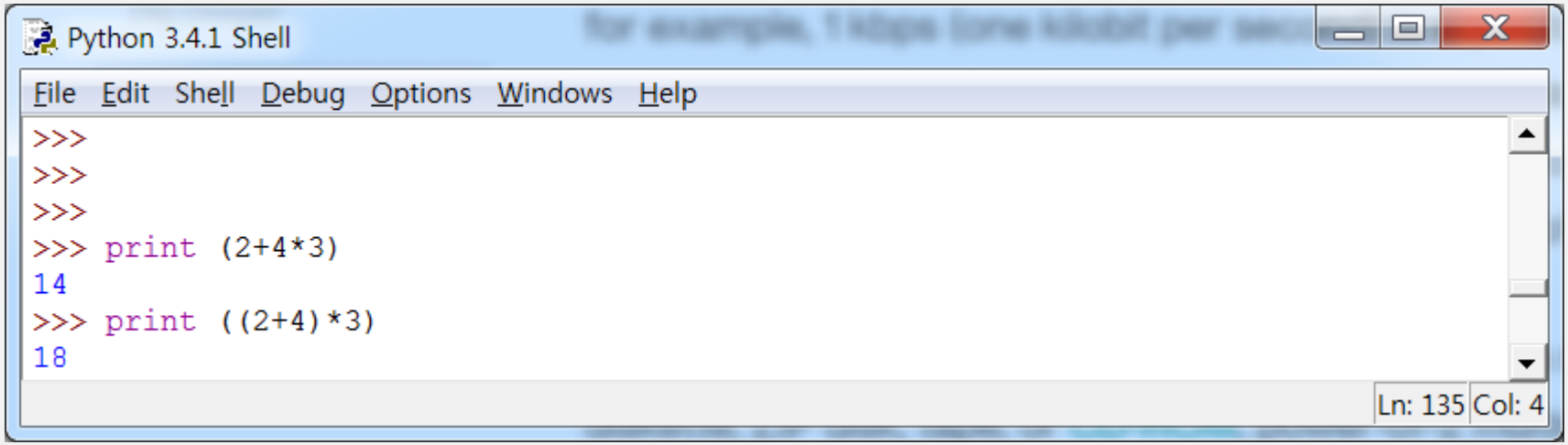


# 대입 연산자

- 다양한 축약 형태 지원
  - +=, -=, *=, /=, **/, //=, %=
  - Ex) x+= 10 → x = x + 10과 동일

# Order of Operations

- 사칙연산 우선 순위 유지
- 괄호
- 기타
  - 줄여 쓰기
    - number +=1 → number = number + 1
    - number /=2 → number = number / 2



# Order of Operations

- 실습
  - 무리수 e를 구하는 코드를 작성하고, 실제 값과 비교해보자.

$$\lim_{x\to\infty}\left(1+\frac{1}{x}\right)^{x}=e$$

```
# 수학과 관련된 모듈
import math

print("실제 값 : ", math.e)
print("x가 10일때 :", (1 + 1/10) ** 10)
print("x 가 100일때 :", (1 + 1/100) ** 100)
print("x 가 1000일때 :", (1 + 1/1000) ** 1000)
print("x 가 10000일때 :", (1 + 1/10000) ** 10000)
```

# 비트 연산자

- Bitwise operator
- 컴퓨터는 정보를 비트로 표시
  - 8비트 → 1바이트, 32/64비트 →1워드(word)

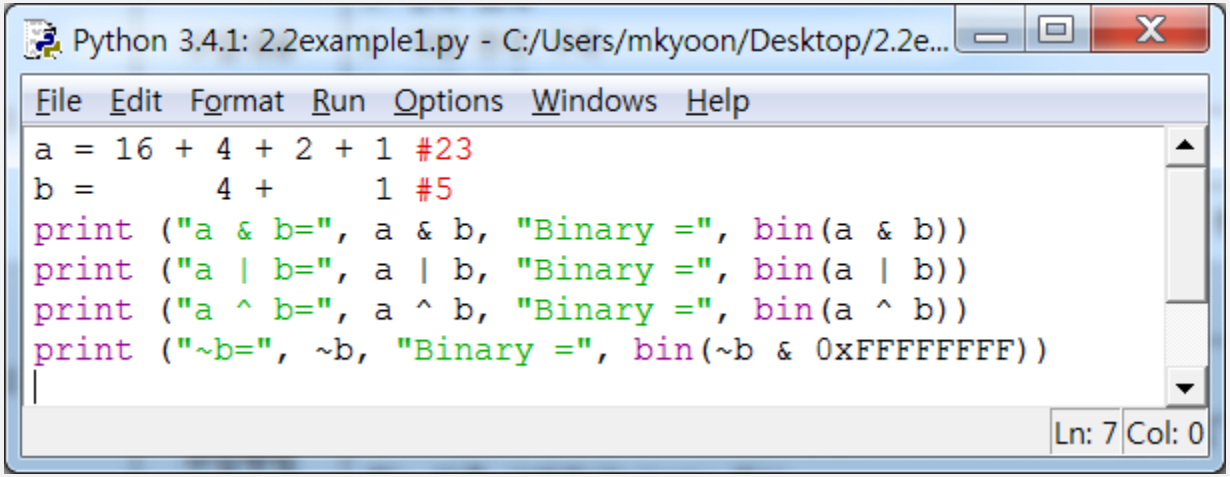
- 비트(이진 데이터) 레벨의 연산자
- 비트 논리 연산자
- 비트 쉬프트(shift) 연산자
- 비트 마스크(mask)

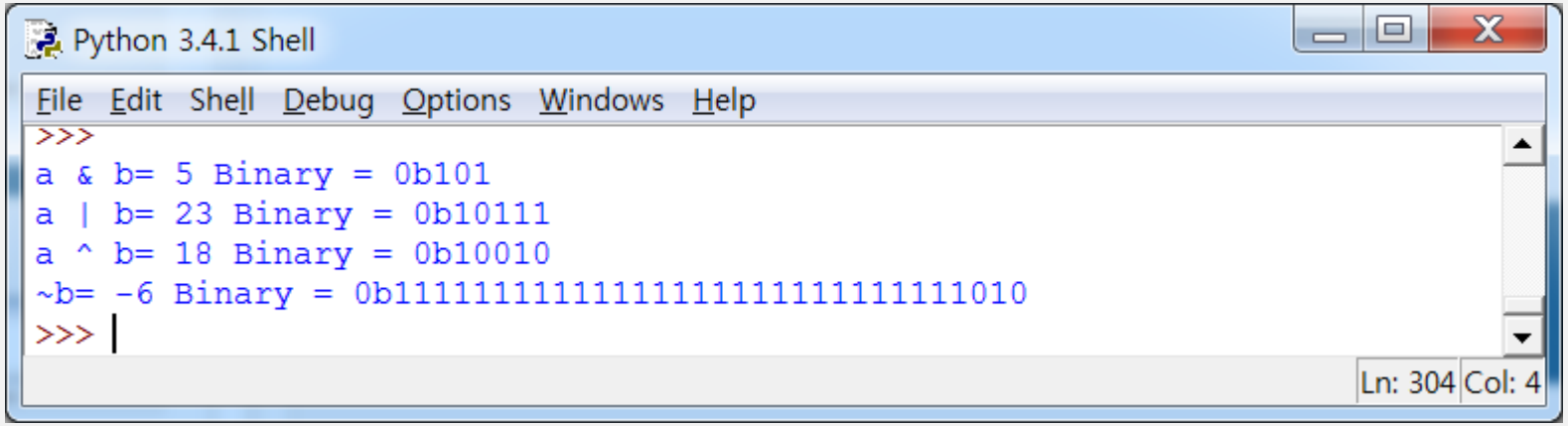
# 비트 연산자

- 비트 논리 연산자
  - &: 비트 논리곱
  - |: 비트 논리합
  - ^: 비트 XOR
    - ^: 캐롯(carrot)
    - XOR: Exclusive OR
  - ~: 비트 논리 부정

# 비트 연산자

- 비트 논리 연산자



# 비트 연산자

- 비트 논리 연산자

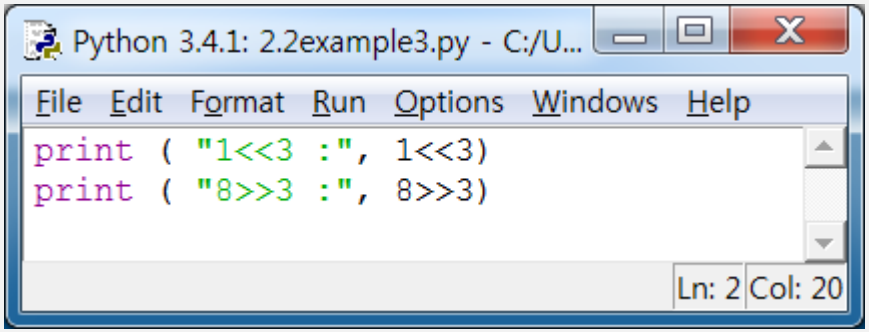
```
# 비트 연산자
# & 는 bit단위에서 같을 때 1, 다를 때 0 이다.
print("23 & 5 :", 23 & 5)
# | 은 bit단위에서 한개라도 1이면 1, 아니면 0이다.
print("23 | 5 :", 23 | 5)
# ^ 은 bit단위에서 두 bit가 다르면 1, 같으면 0이다.
print("23 ^ 5 :", 23 ^ 5)
# a << b 를 하면 a 를 b번만큼 왼쪽으로 bit이동을 시킨다.
print("3 << 3 :", 3 << 3)
# a >> b 를 하면 a 를 b번만큼 오른쪽으로 bit이동을 시킨다.
print("32 >> 2 :", 32 >> 2)
```

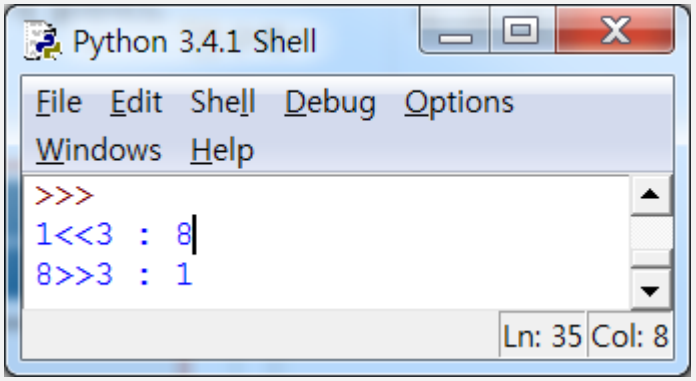
# 비트 연산자

- 비트 논리 연산자
  - ^ (XOR) 연산자 활용 사례 → 암호
    - a와 b가 어떤 값이라도 XOR 연산에 a가 두 번 등장하면 b만 남게됨
      - a ^ b ^ a = b
      - 0 ^ 0 ^ 0 = 0
      - 0 ^ 1 ^ 0 = 1
      - 1 ^ 0 ^ 1 = 0
      - 1 ^ 1 ^ 1 = 1
    - (a ^ b)를 안다고 하더라도, a를 모르면 b를 알아낼 수 없으며, b를 모르면 a를 알아낼 수 없음
      - b: 비밀을 유지하며 친구에게 전달할 문자
      - a: 친구와 사전에 약속한 비밀문자
      - 친구에게 c(=a^b) 전달
      - 친구는 c와 a를 XOR 연산해서 b를 알아냄
      - 다른 사람들은 c를 안다고 하여도 a와 b를 알아내지 못함

# 비트 연산자

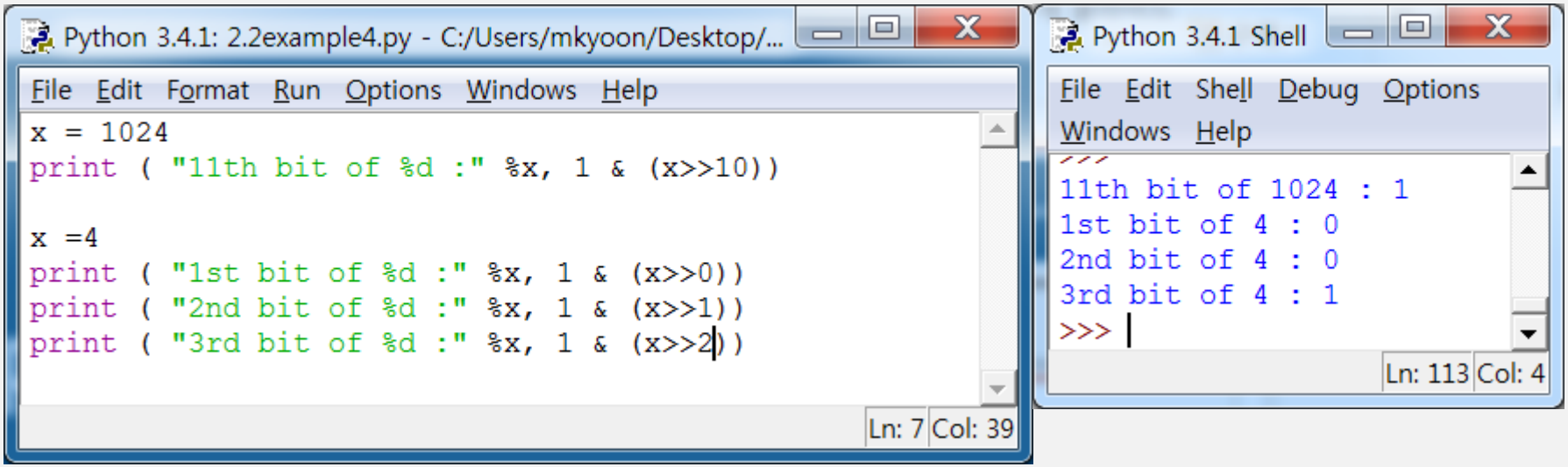
- 비트 쉬프트(shift) 연산자
  - << : 왼쪽으로 비트 이동
    - 1 << 3: 숫자 '1'의 모든 비트를 왼쪽으로 3번 이동시킴. → 8로 변환
    - 숫자를 n번 왼쪽으로 이동시키면 $2^n$을 곱하는 효과와 동일해짐
  - >> : 오른쪽으로 비트 이동
    - 8 >> 3: 숫자 '8'의 모든 비트를 오른쪽으로 3번 이동시킴. → 1로 변환
    - 숫자를 n번 오른쪽으로 이동시키면 $2^n$으로 나누는 효과와 동일해짐



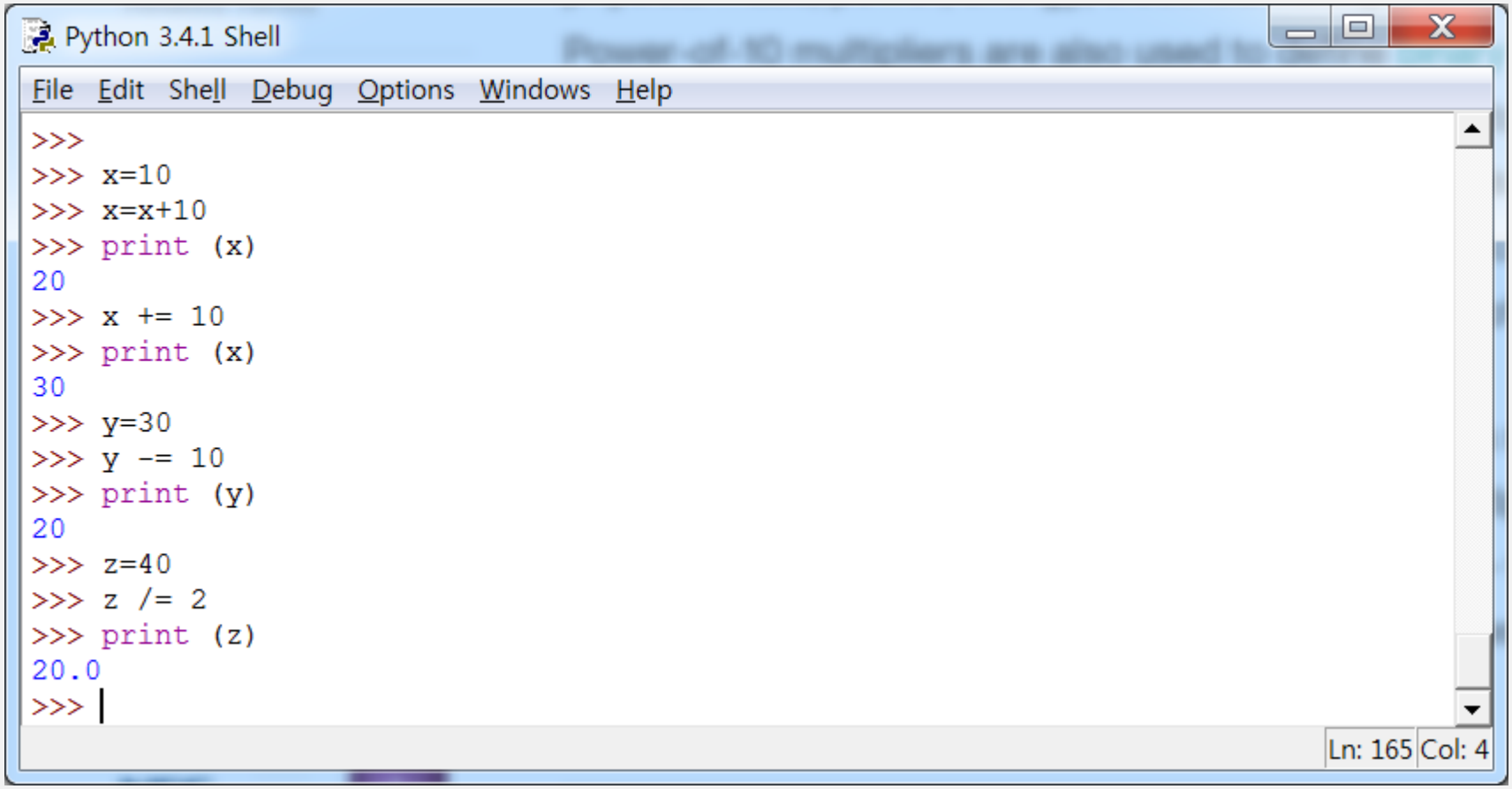
# 비트 연산자

- 비트 마스크(mask)
  - 특정 위치의 비트 값을 알고 싶을 때 사용 (0 or 1?)
  - x의 오른쪽에서부터 n번째 비트 값 확인
    - 1 & (x>>n-1)
    - 연산 후에도 x값은 변화가 없음



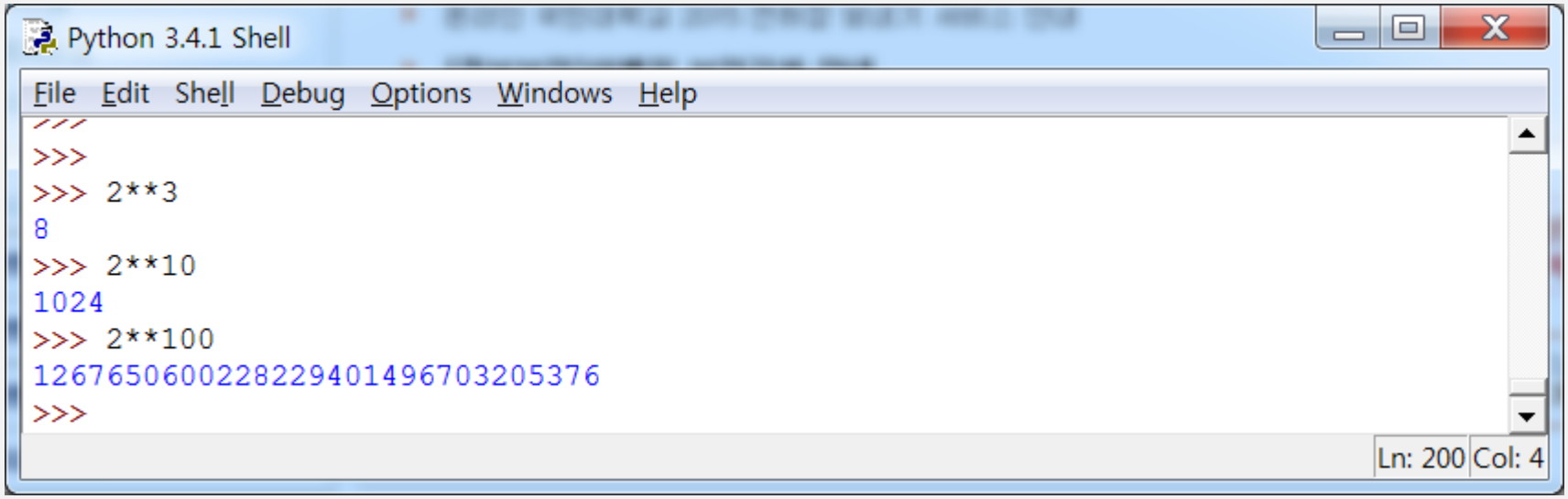
# More Operators

- 증가하기
- 감소하기



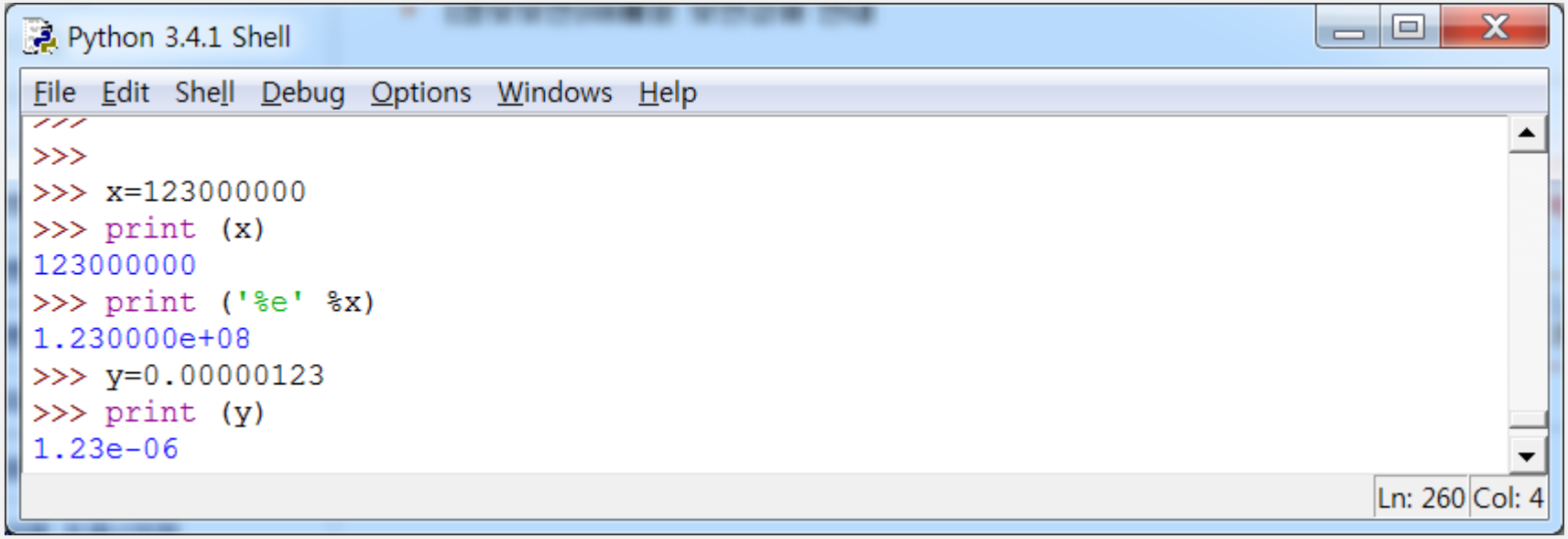
# More Operators

- 지수법
  - 제곱하기
  - **
    - 2**3=8



# Really Big and Really Small

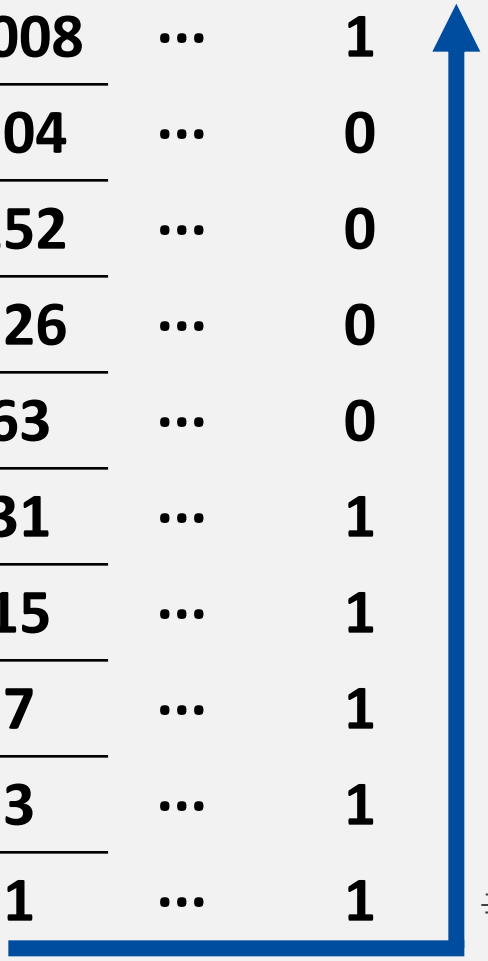
- E-표기법(E-notation)
  - 소수와 10의 제곱승 이용
  - e+08은 10의 8승
  - e-06은 10의 -6승



# 진법 변환

- 10진수 -> 2진수 변환

| | | | |
|---|---|---|---|
| 2 | 2017 | | |
| 2 | 1008 | … | 1 |
| 2 | 504 | … | 0 |
| 2 | 252 | … | 0 |
| 2 | 126 | … | 0 |
| 2 | 63 | … | 0 |
| 2 | 31 | … | 1 |
| 2 | 15 | … | 1 |
| 2 | 7 | … | 1 |
| 2 | 3 | … | 1 |
| | 1 | … | 1 |



컴퓨터에서 2진수를 사용하는 이유?

$$2017_{(10)} = 11111100001_{(2)}$$

# 진법 변환

- 2진수 –> 16진수 변환
  - 4비트의 숫자를 하나로 변환
  - $11111100001_{(2)}=7E1_{(16)}$
- 2진수 –> 8진수 변환
  - 3비트의 숫자를 하나로 변환
  - $11111100001_{(2)}=3741_{(8)}$

| 10진수 | 2진수 | 8진수 | 16진수 |
|---|---|---|---|
| 00 | 0000 | 00 | 0 |
| 01 | 0001 | 01 | 1 |
| 02 | 0010 | 02 | 2 |
| 03 | 0011 | 03 | 3 |
| 04 | 0100 | 04 | 4 |
| 05 | 0101 | 05 | 5 |
| 06 | 0110 | 06 | 6 |
| 07 | 0111 | 07 | 7 |
| 08 | 1000 | 10 | 8 |
| 09 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

# 진법 변환

- 2진수 덧셈

| | | |
|---|---|---|
| | 0111 | 7 |
| + | 0011 | 3 |
| | 1010 | 10 |

- 2진수 뺄셈

| | | |
|---|---|---|
| | 0111 | 7 |
| ? | 0011 | 3 |
| | ? | 4 |

  - 7 – 3이 아닌 7 + ( -3)으로 처리

# 진법 변환

- 음의 10진수 -> 2진수 변환 (MSB 이용)
  - MSB(Most Significant Bit) : 가장 큰 자릿수의 비트
    - 일반적으로 가장 왼쪽 비트
  - MSB가 0일 때 양수, 1일 때 음수
  - $123_{(10)} = 0111\ 1011_{(2)}$
  - $-123_{(10)} = 1111\ 1011_{(2)}$

- 문제

| | | |
|---|---|---|
| | **0001** | **1** |
| **+** | **1001** | **-1** |
| | **1010(10)** | **0** |

- 1의 보수를 사용해서 해결하자

# 진법 변환

- 음의 10진수 -> 2진수 변환 (1의 보수 사용)
  - 1을 0으로 0을 1로 바꿔줌.
  - $123_{(10)} = 0111\ 1011_{(2)}$
  - $-123_{(10)} = 1000\ 0100_{(2)}$

- 문제

| | | |
|---|---|---|
| | **0001** | **1** |
| **+** | **1110** | **-1** |
| | **1111(-0)** | **0** |

  - 0의 표현이 2가지($0000\ 0000_{(2)}(+0)$과 $1111\ 1111_{(2)}(-0)$)

- 1의 보수 대신 2의 보수를 사용하자

# 진법 변환

- 음의 10진수 -> 2진수 변환(2의 보수 사용)
  - 1의 보수 후 1을 더해줌
  - $123_{(10)} = 0111\ 1011_{(2)}$
  - $-123_{(10)} = 1000\ 0101_{(2)}$

- 해결

| | | |
|---|---|---|
| | **0001** | **1** |
| **+** | **1111** | **-1** |
| | **0000** | **0** |

- (8bit 기준) -128~127까지 표현 가능
  - $1000\ 0000_{(2)} \sim 0111\ 1111_{(2)}$

# 진법 변환

- 표현 가능한 범위를 넘으면?

싸이의 세계적인 히트곡 '강남스타일'은 2012년 7월 발표된 이후 현재까지 유튜브 조회수가 21억 건을 돌파하면서 최다 조회수를 기록했다. 이같은 성과는 경이적이다.
1일(현지 시각) 유튜브는 모회사 구글의 SNS(소셜네트워크서비스)인 구글플러스를 통해 강남스타일 조회수가 집계 한계치를 넘어서 집계 시스템을 '업그레이드'해야 했다고 밝혔다. 그러면서 **처음에 유튜브가 설계됐을 때 조회수가 21억4,748만3,647건(유튜브 운영 시스템인 '32 비트 정수'로 표현할 수 있는 최대 조회수)을 넘어서는 동영상이 있을 거라고는 전혀 예상하지 못했다고 덧붙였다.** 유튜브의 맷 맥러논은 회사는 조회수 20억 건이 충분할 것으로 생각했지만, 그렇지 않았다고 언급했다.

http://kr.wsj.com/posts/2014/12/04/%EC%8B%B8%EC%9D%B4-%EA%B0%95%EB%82%A8%EC%8A%A4%ED%83%80%EC%9D%BC-%EB%95%8C%EB%AC%B8%EC%97%90-%EC%9C%A0%ED%88%AC%EB%B8%8C-%EC%A7%91%EA%B3%84-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%97%85%EA%B7%B8%EB%A0%88/

싸이의 '강남스타일' 인기는 상상을 초월합니다. 적어도 유튜브 코드에 의하면 확실히 그랬죠. 유튜브 조회수가 2,147,483,647 (21억) 을 넘어가면서 조회수를 64비트 숫자로 바꾸어 9,223,372,036,854,775,808(922경)까지 평가할 수 있게 바꾸었죠. 유튜브는 왜 조회수를 21억 이상 셀 수 없었던 걸까요?
-중략-
여기서, 유튜브의 가능한 조회수가 2,147,483,647로 4,294,967,295가 아니라는 데 주목할 필요가 있습니다. 이건 왜 다른 걸까요? 32비트가 양수가 아니기 때문이다. 음수를 표기하기 위해서는 절반인 2,147,483,647을 음수에 할당한 것이죠. 유튜브의 32 비트 숫자는 0에서 4,294,967,295까지 표기하는 대신 -2,147,483,648에서 2,147,483,647까지 표기합니다. 유튜브의 조회는 음수가 될 일이 없지만 일반 데이터베이스나 프로그래밍을 할 때는 음수를 써야 될 때가 있습니다. 따라서 첫 자리는 음수인지 양수인지 판별하는 부호로 읽고 32비트 숫자는 -2,147,483,648로 2,147,483,647 읽는 것이 컴퓨터의 표준 언어지요. 유튜브의 조회수가 20억을 넘어가면서 32비트가 표현할 수 있는 최고 숫자를 넘어가자 더 이상 컴퓨터는 제대로 숫자를 읽을 수가 없었습니다. 결국 유튜브는 조회수를 64비트로 만들고 9,223,372,036,854,775,808(922경)까지 읽게 변환하였죠. (http://newspeppermint.com/2014/12/16/binary-bug/)

# 진법 변환

- 10진수 -> 2진수 변환 : bin(int)

**bin(*x*)**

Convert an integer number to a binary string. The result is a valid Python expression. If *x* is not a Python `int` object, it has to define an `__index__()` method that returns an integer.

- 10진수 -> 8진수 변환 : oct(int)

**oct(*x*)**

Convert an integer number to an octal string. The result is a valid Python expression. If *x* is not a Python `int` object, it has to define an `__index__()` method that returns an integer.

# 진법 변환

- 10진수 -> 16진수 변환 : hex(int)

**hex(*x*)**

Convert an integer number to a lowercase hexadecimal string prefixed with "0x", for example:

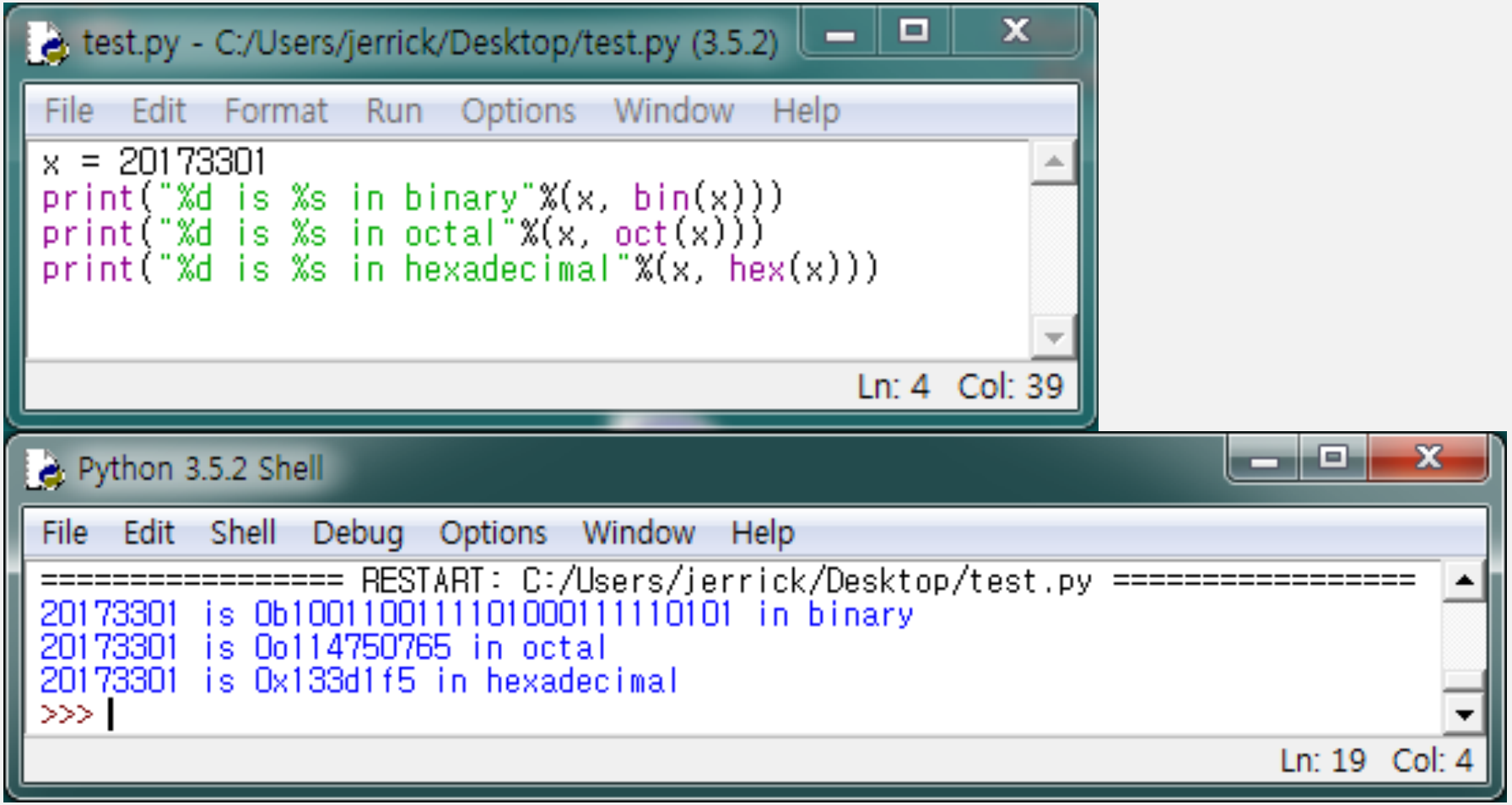
```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

If x is not a Python `int` object, it has to define an __index__() method that returns an integer.

See also `int()` for converting a hexadecimal string to an integer using a base of 16.

# 진법 변환

- 10진수 -> 2, 8, 16진수 변환



# 진법 변환

- N진수 -> 10진수 변환($2 \leq n \leq 32$)

*class* **int** (*x=0*)

*class* **int** (*x*, *base=10*)

Return an integer object constructed from a number or string *x*, or return `0` if no arguments are given. If *x* is a number, return `x.__int__()`. For floating point numbers, this truncates towards zero.

If *x* is not a number or if *base* is given, then *x* must be a string, `bytes`, or `bytearray` instance representing an integer literal in radix *base*. Optionally, the literal can be preceded by `+` or `-` (with no space in between) and surrounded by whitespace. A base-n literal consists of the digits 0 to n-1, with `a` to `z` (or `A` to `Z`) having values 10 to 35. The default *base* is 10. The allowed values are 0 and 2–36. Base-2, -8, and -16 literals can be optionally prefixed with `0b/0B`, `0o/0O`, or `0x/0X`, as with integer literals in code. Base 0 means to interpret exactly as a code literal, so that the actual base is 2, 8, 10, or 16, and so that `int('010', 0)` is not legal, while `int('010')` is, as well as `int('010', 8)`.

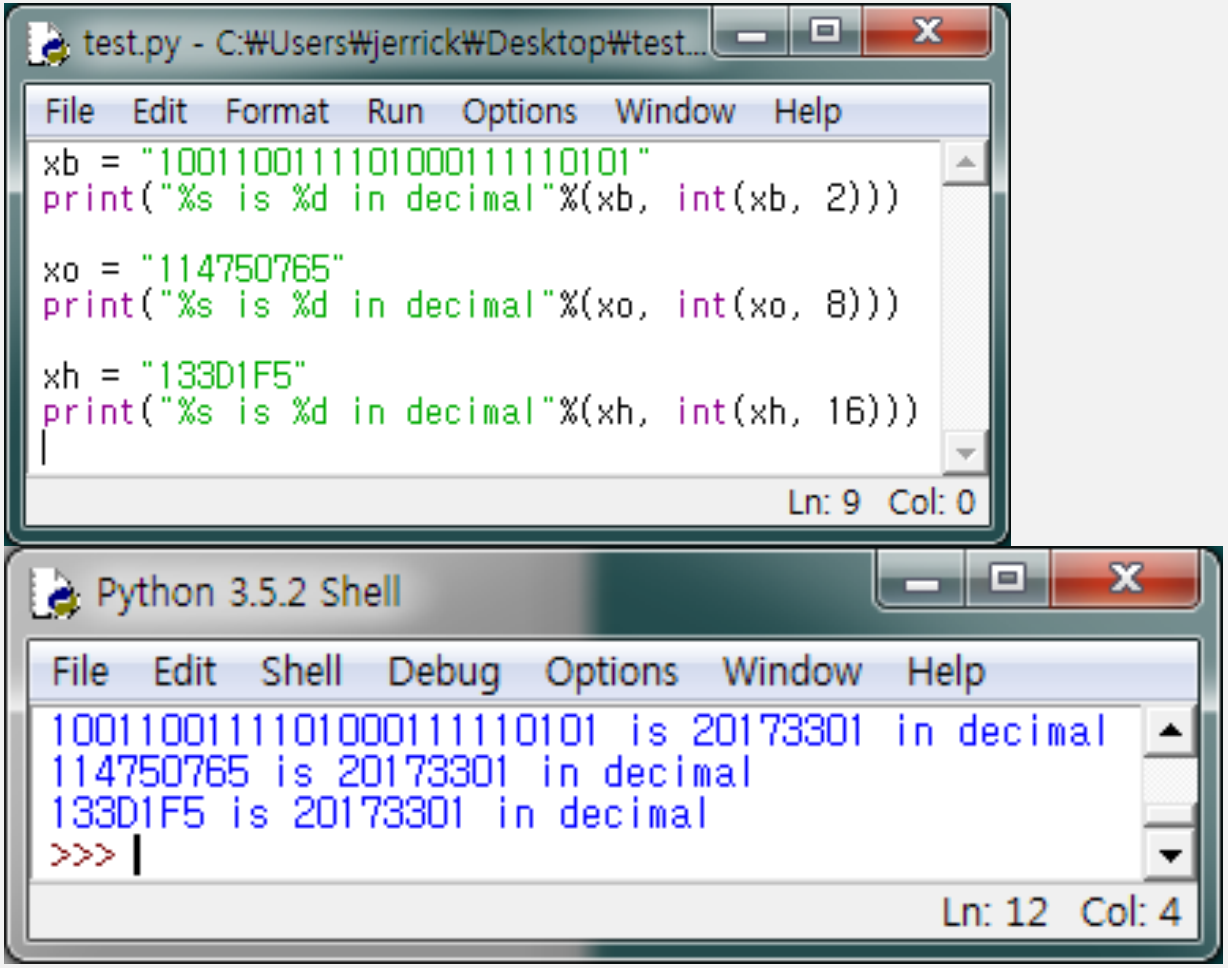
The integer type is described in Numeric Types — int, float, complex.

*Changed in version 3.4:* If *base* is not an instance of `int` and the *base* object has a `base.__index__` method, that method is called to obtain an integer for the base. Previous versions used `base.__int__` instead of `base.__index__`.

*Changed in version 3.6:* Grouping digits with underscores as in code literals is allowed.

# 진법 변환

- N진수 -> 10진수 변환($2 \leq n \leq 32$)



# 심화학습

- 비트(bit)
  - 컴퓨터의 기본 정보 단위
    - `0'과 `1' 표현
    - 전기가 들어오면 `1', 나가면 `0'
  - 3비트로 표현할 수 있는 정보양
    - 000, 001, 010, 011, 100, 101, 110, 111 → 8개
    - 정보양 → 구분할 수 있는 아이템 개수
  - N비트 → $2^n$ 표현 가능 (0 ~ $2^n$-1)
  - 8비트 = 1 바이트(byte)
    - 0-255 표현 가능
  - 32비트 운영 체제: $2^{32}$ 인식
  - 64비트 운영 체제: $2^{64}$ 인식

# 심화학습

- 2의 지수승
  - 1 바이트(byte) → $2^{32}$
  - 10비트 → $2^{10}$ = 1,024 ≈ 1,000 (Kilo)
  - 20비트 → $2^{20}$ = $(2^{10})^2$=1,048,576 ≈ 1,000,000 = 1 Mega
  - 30비트 → $2^{30}$ = $(2^{10})^3$=1,073,741,824 ≈ 1,000,000,000 = 1 Giga
  - 32비트 → $2^{32}$ = $(2^{10})^3 \times 2^2$ = 4,294,967,296 ≈ 4,000,000,000 = 4 Giga
- 32비트 윈도우7 운영체제는 최대 4GB(Giga Byte)의 메모리만 장착할 수 있다. 정확히는 더 큰 메모리를 장착해도 4GB만 인식하여 사용할 수 있다. 그 이유에 대해서 생각해 보시오.

- 64비트로 표현 가능한 가장 큰 숫자를 Python “print”를 사용해서 계산하시오.

# 심화 학습

- 국제단위계-SI접두어

| $10^n$ | 접두어 | 기호 | 배수 |
|---|---|---|---|
| $10^{24}$ | 요타 (yotta) | Y | 자 |
| $10^{21}$ | 제타 (zetta) | Z | 십해 |
| $10^{18}$ | 엑사 (exa) | E | 백경 |
| $10^{15}$ | 페타 (peta) | P | 천조 |
| $10^{12}$ | 테라 (tera) | T | 조 |
| $10^{9}$ | 기가 (giga) | G | 십억 |
| $10^{6}$ | 메가 (mega) | M | 백만 |
| $10^{3}$ | 킬로 (kilo) | k | 천 |
| $10^{2}$ | 헥토 (hecto) | h | 백 |
| $10^{1}$ | 데카 (deca) | da | 십 |
| $10^{0}$ | | | 일 |

| $10^n$ | 접두어 | 기호 | 배수 |
|---|---|---|---|
| $10^{0}$ | | | 일 |
| $10^{-1}$ | 데시 (deci) | d | 십분의 일 |
| $10^{-2}$ | 센티 (centi) | c | 백분의 일 |
| $10^{-3}$ | 밀리 (milli) | m | 천분의 일 |
| $10^{-6}$ | 마이크로 (micro) | μ | 백만분의 일 |
| $10^{-9}$ | 나노 (nano) | n | 십억분의 일 |
| $10^{-12}$ | 피코 (pico) | p | 일조분의 일 |
| $10^{-15}$ | 펨토 (femto) | f | 천조분의 일 |
| $10^{-18}$ | 아토 (atto) | a | 백경분의 일 |
| $10^{-21}$ | 젭토 (zepto) | z | 십해분의 일 |
| $10^{-24}$ | 욕토 (yocto) | y | 일자분의 일 |

# 실습

- 오차율 계산
  - 10비트로 표현하는 kilo($2^{10}$ = 1,024)와 실제 1,000과의 오차율을 계산해라.
    - ($2^{10}$-1000)/1000
  - 20비트에 대해서도 오차율을 계산하라.
  - 30비트에 대해서도 오차율을 계산하라.

# 실습

- 나머지 연산(modular)은 그룹과 패턴 관련 문제 풀이에서 자주 등장함
- 요일 예측
  - 오늘이 일요일이면, 100일 후는 무슨 요일인가?
    - 1주일=7일 → 7, 14, 21, 28,…, 98일 후는 일요일. 100일 후는 화요일
    - 7로 나누었을 때 나머지
      - 0: 일요일
      - 1: 월요일
      - 2: 화요일
      - 3: 수요일
      - 4: 목요일
      - 5: 금요일
      - 6: 토요일
    - 1억일(100,000,000) 후 요일을 Python으로 계산하시오.