Programming in Python

05 – 수치 자료형과 연산

2016년 8월, 국민대학교 컴퓨터공학부

수치형 상수 (Numeric Literals)

Literal	Interpretation
1234, -24, 0, 99999999999999	정수형 (크기 제한 없음)
1.23, 1., 3.14e-10, 4E210, 4.0e+210	부동소수점 표현
0177, 0x9ff, 0b101010	8진수, 16진수, 2진수 (Python 2.6)
0o177, 0x9ff, 0b101010	8진수, 16진수, 2진수 (Python 3.0)
3+4j, 3.0+4.0j, 3J	복소수 표현

- 정수형에 대해서 표현할 수 있는 수의 범위 제한 없음
  - Python 2.6 에서는 32-bit 정수와 제한 없는 정수 표현을 자동으로 이용
- 실수 (부동소수점) 에 대해서 최대 정밀도는 Python 시스템을 빌드한 C 컴파일러에 의존
  - “double” precision in C
- 복소수 표현에 대해서 별도의 프로그래밍 없이 지원

Python 에서의 나눗셈

Python 3.0

```
>>> 10 / 4
2.5
>>> 10 // 4
2
>>> 10 / 4.0
2.5
>>> 10 // 4.0
2.0
```

Python 2.6

```
>>> 10 / 4
2
>>> 10 // 4
2
>>> 10 / 4.0
2.5
>>> 10 // 4.0
2.0
```

Python 2.x 와 Python 3.x 에 차이가 있음

이런 차이를 알아 두어야 할까?

이런 차이 때문에 쓸 데 없이
고생할 가능성이 상당히 있음

Python 2.6

```
>>> from __future__ import division
>>> 10 / 4
2.5
>>> 10 // 4
2
```

Python 에서의 나눗셈

Python 3.0

```
>>> 5 / 2, 5 / -2
(2.5, -2.5)

>>> 5 // 2, 5 // -2
(2, -3)

>>> 5 / 2.0, 5 / -2.0
(2.5, -2.5)

>>> 5 // 2.0, 5 // -2.0
(2.0, -3.0)
```

Python 2.6

```
>>> 5 / 2, 5 / -2
(2, -3)

>>> 5 // 2, 5 // -2
(2, -3)

>>> 5 / 2.0, 5 / -2.0
(2.5, -2.5)

>>> 5 // 2.0, 5 // -2.0
(2.0, -3.0)
```

```
>>> import math
>>> math.floor(2.5)
2

>>> math.floor(-2.5)
-3

>>> math.trunc(2.5)
2

>>> math.trunc(-2.5)
-2
```

Python 이 가지는 수치형의 편리함

- 큰 정수 표현

```
>>> 2 ** 200
1606938044258990275541962092341162602522202993782792835301376L
```

- 복소수 지원

```
>>> 1j * 1J
(-1+0j)
```

실수부와 허수부는 공히 부동소수점 소수로 취급됨

```
>>> 2 + 1j * 3
(2+3j)

>>> (2 + 1j) * 3
(6+3j)
```

16진수, 8진수, 2진수 표현

정수형 상수 표현의 다른 형태일 뿐

```
>>> 0o1, 0o20, 0o377
(1, 16, 255)

>>> 0x01, 0x10, 0xFF
(1, 16, 255)

>>> 0b1, 0b10000, 0b11111111
(1, 16, 255)
```

```
>>> oct(64)
'0o100'

>>> hex(64)
'0x40'

>>> bin(64)
'0b1000000'
```

각각 8진수, 16진수, 2진수 표현의 문자열이 생성됨

```
>>> int('64'), int('100', 8), int('40', 16), int('1000000', 2)
(64, 64, 64, 64)
```

문자열 → 각 진법으로 표현된 정수

```
>>> int('0x40', 16), int('0b1000000', 2)
(64, 64)
```

문자열은 Python 상수 표현도 가능

```
>>> eval('64'), eval('0o100'), eval('0x40'), eval('0b1000000')
(64, 64, 64, 64)
```

문자열을 Python 코드로 간주

비트 단위의 연산 (Bitwise Operations)

```
>>> X = 0b0001                  # 2진 상수 표현
>>> X << 2                      # 왼쪽 시프트 (2 자리)
4
>>> bin(X << 2)                 # 2진 문자열 표현으로 하면?
'0b100'
>>> bin(X | 0b010)              # 비트 수준의 OR
'0b11'
>>> bin(X & 0b1)                # 비트 수준의 AND
'0b1'

>>> X = 0xFF                    # 16진 상수 표현
>>> bin(X)                      # 2진 문자열 표현으로 하면?
'0b11111111'
>>> X ^ 0b10101010              # 비트 수준의 XOR
85
>>> bin(X ^ 0b10101010)         # 2진 문자열 표현으로 하면?
'0b1010101'
```

집합 (Sets)

집합 (set): 다른 Python 객체들 (objects) 의 집합 (순서 없음)

```
>>> x = set('abcde')                    # 문자열을 이용한 집합
>>> y = set('bdxyz')                    # → 각 문자가 집합의 원소가 됨
>>> x
set(['a', 'c', 'b', 'e', 'd'])          # Python 2.6
또는, {'a', 'c', 'b', 'e', 'd'}                    # Python 3.0

>>> 'e' in x                            # 원소의 포함 관계
True
>>> x - y                               # 차집합
set(['a', 'c', 'e'])
>>> x | y                               # 합집합
set(['a', 'c', 'e', 'b', 'd', 'y', 'x', 'z'])
>>> x & y                               # 교집합
set(['b', 'd'])
>>> x ^ y                               # XOR
set(['a', 'c', 'e', 'y', 'x', 'z'])
>>> x > y, x < y                        # 집합의 포함 관계
(False, False)
```

집합 내포 (Set Comprehension)

```
>>> {x ** 2 for x in [1, 2, 3, 4]}
{16, 1, 4, 9}

>>> {x for x in 'spam'}                    # set('spam') 과 동일
{'a', 'p', 's', 'm'}

>>> {c * 4 for c in 'spam'}
{'ssss', 'aaaa', 'pppp', 'mmmm'}

>>> S = {c * 4 for c in 'spam'}

>>> S | {'mmmm', 'xxxx'}
{'ssss', 'aaaa', 'pppp', 'mmmm', 'xxxx'}

>>> S & {'mmmm', 'xxxx'}
{'mmmm'}
```

집합의 이용

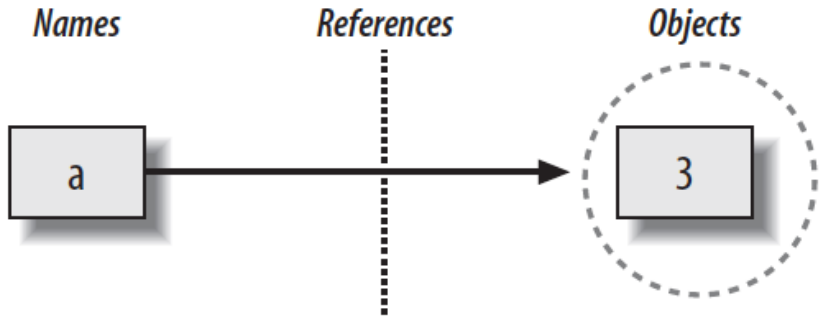
```
>>> engineers = {'bob', 'sue', 'ann', 'vic'}
>>> managers = {'tom', 'sue'}

>>> 'bob' in engineers                  # Is bob an engineer?
True

>>> engineers & managers                # Who is both engineer and manager?
{'sue'}

>>> engineers | managers                # All people in either category
{'vic', 'sue', 'tom', 'bob', 'ann'}

>>> engineers - managers                # Engineers who are not managers
{'vic', 'bob', 'ann'}

>>> managers - engineers                # Managers who are not engineers
{'tom'}

>>> {'bob', 'sue'} < engineers          # Are both engineers? (subset)
True
```

Python 변수와 저장 공간

- 변수는 처음으로 값이 대입될 때 객체로서 생성된다.
  - 이 변수의 자료형은 Python 이 자동으로 결정한다.
- 변수는 참조될 때 (수식에 포함될 때) 그 내용 (값) 으로 대체된다.
- 변수는 참조되기 이전에 생성되어 있어야 한다.
  - 그렇지 않은 경우는 에러
- 변수는 객체를 가리키고 있으며, 미리 선언되지 않는다.

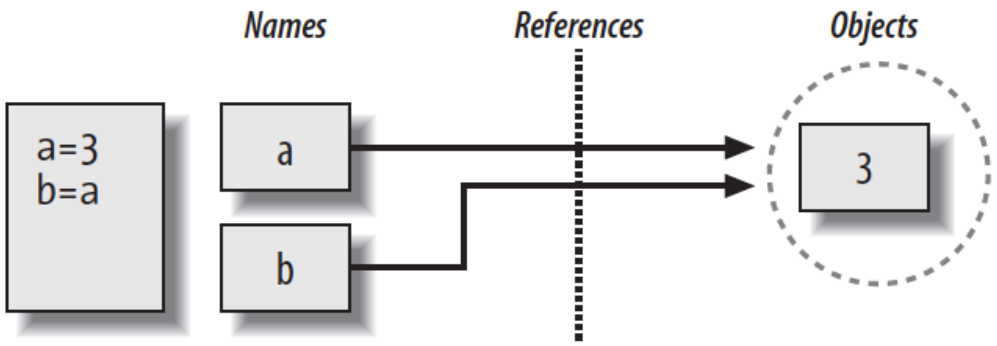
```
>>> a = 3
```

1. 값 (정수형, 3) 을 가지는 객체를 생성한다.
2. 이름 (a) 을 가지는 변수를 생성한다.
   (이미 생성되어 있었던 경우가 아니라면)
1. 변수 a 를 1 에서 생성된 객체를 가리키도록 한다.

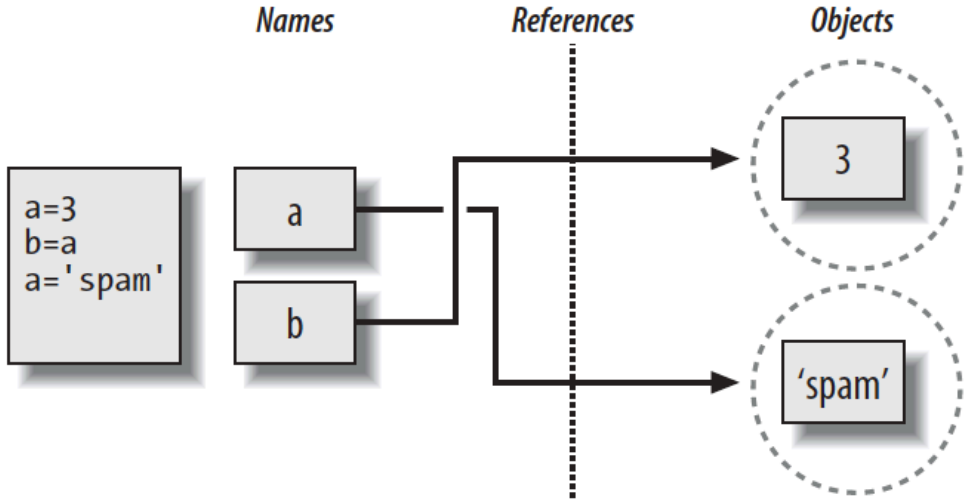


참조의 공유 (Shared References)

```
>>> a = 3
>>> b = a
```




```
>>> a = 3
>>> b = a
>>> a = 'spam'
```



공유된 참조와 객체의 변화

```
>>> L1 = [2, 3, 4]      # 리스트는 변경 가능한 객체
>>> L2 = L1             # 참조 공유
>>> L1[0] = 24          # 객체에 변화가 발생

>>> L1                  # L1 에는 변경사항이 생겼음
[24, 3, 4]
>>> L2                  # L2 에도 같은 변경사항!
[24, 3, 4]
```

```
>>> L1 = [2, 3, 4]
>>> L2 = L1[:]          # L1 의 복사본을 만들었음
>>> L1[0] = 24

>>> L1                  # L1 에는 변경사항이 생겼음
[24, 3, 4]
>>> L2                  # L2 에는 변화 없음
[2, 3, 4]
```

공유된 참조와 등치

```
>>> L = [1, 2, 3]
>>> M = L                # M 과 L 은 같은 객체를 가리키고 있음
>>> L == M               # 값이 같은지? (양변은 각각 계산됨)
True
>>> L is M               # 같은 객체인지?
True
```

```
>>> L = [1, 2, 3]
>>> M = [1, 2, 3]
>>> L == M               # 값이 같은지?
True
>>> L is M               # 같은 객체인지?
False
```

```
>>> X = 42
>>> Y = 42
>>> X == Y               # 값이 같은지?
True
>>> X is Y               # 같은 객체인지?
True
```

정수 42 에 해당하는 객체가 생성되고
그 객체가 캐싱 (caching) 되어 참조되기 때문

Quiz

- 아래 각각의 실행 결과는?
  - 그리고, 각각 그 이유는?

```
>>> A = "spam"
>>> B = A
>>> B = "shrubbery"
>>> A
???
```

```
>>> A = ["spam"]
>>> B = A
>>> B[0] = "shrubbery"
>>> A
???
```

```
>>> A = "spam"
>>> B = A[:]
>>> B[0]  = "shrubbery"
>>> A
???
```

- 비교

```
>>> A = "spam"
>>> B = A
>>> B[0] = "x"
???
```

Exercise

(1) 2부터 30 까지의 소수를 모두 찾아내는 프로그램을 작성하시오.

A. 결과는 리스트로 만들어 출력할 것

B. 정답은: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

Answer

```
primes = []

for i in range(2, 31):
    bPrime = True
    for j in range(2, i // 2 + 1):
        if i % j == 0:
            bPrime = False
    if bPrime == True:
        primes.append(i)

print primes
```

```
import math

primes = []

for i in range(2, 31):
    bPrime = True
    for j in range(2, int(math.sqrt(i)) + 1):
        if i % j == 0:
            bPrime = False
    if bPrime == True:
        primes.append(i)

print primes
```

Q & A