Python 과학 프로그래밍 기초
Python 문법 기초 (3)

한림대학교 소프트웨어융합대학
박섭형

2021년 1학기

2. Python 문법 기초 (3)

- 정수의 비트 단위 연산
- 할당 연산자
- 실수형 자료의 표현
- 명령줄과 들여쓰기
- 주석문

연산자	의미
&	비트단위 AND
|	비트단위 OR
~	비트단위 NOT
^	비트단위 XOR
≫	비트단위 우측 이동
≪	비트단위 좌측 이동

정수의 비트 단위 연산 (Bit-wise operation)

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1

a	b	a & b	a | b	a ^ b
1	1	1	1	0

1 비트 and / or / xor

Python 정수의 bit_length

[1]:
```
m = 0b1101
n = 0b110
m, n
```

[1]:
```
(13, 6)
```

[2]:
```
m.bit_length(),(-m).bit_length(),  n.bit_length(), (-n).bit_length()
```

[2]:
```
(4, 4, 3, 3)
```

정수의 비트단위 not 연산

~m은 m의 1의 보수(1's complement)

- 13_{10}과 $\sim 13_{10}$

$$13_{10} = 01101_2 (13_{10})$$

$$\sim 13_{10} = 0\ 1101_2 \text{ 의 1의 보수} = 1\ 0010_2$$

음수는 2의 보수(2's complement)로 표현된다

- 13_{10}과 -13_{10}
  - $13_{10} = 0\ 1101_2 (13_{10})$
  - $0\ 1101_2$의 1의 보수 = $1\ 0010_2$ (-14_{10})
  - $0\ 1101_2$의 2의 보수 = $1\ 0011_2$ (-13_{10})

[3]:
```
-m == ~m + 1
```

[3]:
```
True
```

[4]:
```
~m == -m - 1
```

[4]: True

[5]:
```
m, ~m, -m, ~-m
```

[5]: (13, -14, -13, 12)

	이진수	십진수
m	0 1101	13
n	0 0110	6
m & n	0 0100	4
m | n	01111	15
m ^ n	01011	11

정수의 비트 단위 and / or / xor 예

[6]:
```
m&n, m|n, m^n
```

[6]: (4, 15, 11)

	이진수	십진수
−m	1 0011	−13
−n	1 1010	−6
−m & −n	1 0010	−14
−m | −n	1 1011	−5
−m ^ −n	0 1001	9

음수 정수의 비트단위 연산

[7]:
```
-m & -n, -m|-n, (-m)^(-n)
```

[7]: (-14, -5, 9)

[8]:
```
-m & n, m & -n, -m & -n
```

[8]: (2, 8, -14)

[9]:
```
-m & -n, -m|-n, (-m)^(-n)
```

[9]: (-14, -5, 9)

정수의 비트 단위 이동 (Bit-wise shift)

- m ≫ n은 m//(2**n)과 동일
- m ≪ n은 m*(2**n)과 동일

[10]:
```
15 >> 2, 15 // 2**2, 3877 >> 5, 3877 // 2**5
```

[10]:
```
(3, 3, 121, 121)
```

[11]:
```
-15 >> 2, -15 // 2**2, -3877 >> 5, -3877 // 2**5
```

[11]:
```
(-4, -4, -122, -122)
```

m(십진수)	m(이진수)	m ≫ 1 (이진수, 10진수)	m ≫ 2 (이진수, 10진수)	m // 4 (십진수)
15	01111	00111 (7)	00011 (3)	3
−15	10001	11000 (−8)	11100 (−4)	−4

$$15 = (3)(4) + 3$$

$$-15 = (-4)(4) + 1$$

할당 연산자

연산	예	동치 연산
=	x = 3	
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
//=	x //= 3	x = x // 3
%=	x %= 3	x = x % 3
=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
^=	x ^= 3	x = x ^ 3
|=	x |= 3	x = x | 3
≫=	x ≫= 3	x = x ≫ 3
≪=	x ≪= 3	x = x ≪ 3

실수 자료형의 표현

- 디지털 컴퓨터는 2진수를 사용
- 범위(range) vs 정밀도(precision)
- 고정 소숫점(fixed-point) 실수 표현
- 부동소숫점(floating-point) 실수 표현
  - IEEE 754 표준
  - 근사값 표현

실수 자료형 연산 예

- $5/3 = 1.66666\cdots = 1.\overline{6}$
- $8/3 = 2.66666\cdots = 2.\overline{6}$
- $8/3 - 1 = 5/3$

[12]:
```
5/3, 8/3, 8/3 - 5/3
```

[12]:
```
(1.6666666666666667, 2.6666666666666665, 0.9999999999999998)
```

[13]:
```
8/3 - 5/3 == 1
```

[13]:
```
False
```

[14]:
```
0.1 + 0.1 + 0.1 == 0.3
```

[14]:
```
False
```

[15]:
```
0.1
```

[15]:
```
0.1
```

[16]:
```
0.1 + 0.1
```

[16]:
```
0.2
```

[17]:
```
0.1 + 0.1 + 0.1
```

[17]:
```
0.30000000000000004
```

[18]:
```
print(0.3)
print(f"{0.3:.18f}")
```

```
0.3
0.299999999999999989
```

실수의 부동 소수점 표현

- 디지털 컴퓨터는 모든 실수를 표현하지 못한다.
- 실수의 집합 R은 부동 소수점 수의 집합으로 근사화된다.
- 부동 소수점 수의 집합 F는 다음과 같이 표현된다.

$$F = \left\{ x \mid x = \pm \left(d_0 + \frac{d_1}{\beta^1} + \frac{d_2}{\beta^2} + \cdots + \frac{d_{p-1}}{\beta^{p-1}} \right) \times \beta^e \right\},$$

- β: 밑수(base), $\beta = 2, 10, 16$
- p: 정밀도(precision)
- e: 지수(exponent). $L \leq e \leq U$ 인 정수이며, $[L, U]$를 지수의 범위라고 함.
- 문자열 $d_0 d_1 d_2 \cdots d_{p-1}$을 가수(fraction, mantissa)라고 한다.

부동 소수점 수

- 밑수(base)가 10인 부동 소수점 수
- $0.0005415 = 5.415 \times 10^{-4} = \left(5 + \frac{4}{10} + \frac{1}{10^2} + \frac{5}{10^3} \right) 10^{-4}$
- $-120.15 = -1.2015 \times 10^2 = -\left(1 + \frac{2}{10} + \frac{0}{10^2} + \frac{1}{10^3} + \frac{5}{10^4} \right) 10^2$

$$x = \pm \left(d_0 + \frac{d_1}{10^1} + \frac{d_2}{10^2} + \cdots + \frac{d_{p-1}}{10^{p-1}} \right) \times 10^e$$

- 5.5625를 밑수(base)가 2인 부동 소수점 수로 표현하는 경우
- $101.1001 = 1.011001 \times 2^2 = \left(1 + \frac{0}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{0}{2^4} + \frac{0}{2^5} + \frac{1}{2^6} \right) 2^2$

$$x = \pm \left(d_0 + \frac{d_1}{2^1} + \frac{d_2}{2^2} + \cdots + \frac{d_{p-1}}{2^{p-1}} \right) \times 2^e$$

IEEE 754 부동 소수점 표현

- 16-bit: Half (binary16)
- 32-bit: Single (binary32), decimal32
- 64-bit: Double (binary64), decimal64
- 128-bit: Quadruple (binary128), decimal128
- 256-bit: Octuple (binary256)
- 40-bit or 80-bit: Extended precision
- 기타

[61]:
```
import sys
sys.float_info
```

[61]:
```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15,
mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

IEEE 754 표준 배정도 부동 소수점 표현

- 기본 형식 : $x = (-1)^s \times c \times \beta^e$
- 배정도(double precision) 부동 소수점: binary64
  - $\beta = 2$
  - s: 1 비트 길이의 부호 비트, b_{63}
  - q: 11 비트 부호 없는 정수 (unsigned int), $q = e + (2^10 - 1) = e + 1023$

    $b_{62}b_{61}\cdots b_{52}$

    $e = q - 1023, \;\; 1 \le q \le 2046 \;\; 000_{16}$ 과 $7FF_{16}$ 제외 $-1023 \le e \le 1023$
  - c: d_0를 제외한 52 비트 길이의 가수, $b_{51}b_{50}\cdots b_1b_0$

- $0\ 00000000000\ 0000\cdots000_2$: 0
- $1\ 00000000000\ 0000\cdots000_2$: -0
- $0\ 11111111111\ 0000\cdots000_2$: +infinity
- $1\ 11111111111\ 0000\cdots000_2$: $-$infinity
- $0\ 11111111111\ 0000\cdots001_2$: NaN
- $0\ 11111111111\ 1000\cdots001_2$: NaN
- $0\ 11111111111\ 1111\cdots111_2$: NaN

- IEEE-754 배정도 부동 소수점 형식의 이진 문자열 $b_{63}b_{62}b_{61}\cdots b_2b_1b_0$를 10 진수 x로 변환하면

$$
\begin{aligned}
x &= (-1)^{b_{63}} \times (1.d_{51}d_{50}\cdots d_1d_0)_2 \times 2^{(b_{62}b_{61}\cdots b_{52})_2 - 1023} \\
&= (-1)^{b_{63}} \left(1 + \sum_{i=1}^{51} b_{52-i}2^{-i}\right) \times 2^{q-1023}, \\
q &= \sum_{k=0}^{10} b_{k+52}2^k
\end{aligned}
$$

- 유효자리 숫자
  - 2진수 53자리
  - 10진수: $\log_{10}(2^{53} = 15.95$

[62]:
```
for n in range(30):
    print(n%10, end='')
print()
print(1/8)
print(1/3)
print(1/33)
```

```
012345678901234567890123456789
0.125
0.3333333333333333
0.030303030303030304
```

[64]:
```
for n in range(30):
    print(n%10, end='')
print()
x = 1/33
print(x)
s1 = f'{x:0.15f}'
s2 = f'{x:0.16f}'
s3 = f'{x:0.17f}'
s4 = f'{x:0.18f}'
s5 = f'{x:0.19f}'
print(eval(s1)==x, eval(s2)==x, eval(s3)==x, eval(s4)==x, eval(s5)==x)
```

```
012345678901234567890123456789
0.030303030303030304
False False False True True
```

floationg point reference sheet for intel architecture

10진수 14.40625를 2진수로 변환하기

- $14_{10} = 1110_2$

[65]:
```
print(bin(14), bin(14)[2:])
```

```
0b1110 1110
```

- $0.40625_{10} = 0.01101_2$

$$14.40625_{10} = 1110.01101_2 = 1.11001101_2 \times 2^3$$

[66]:
```
(14.40625).hex()
```

[66]:
```
'0x1.cd00000000000p+3'
```

[67]:
```
bin(0xcd)
```

[67]:
```
'0b11001101'
```

[70]:
```
import math
math.isclose(0.1+0.1+0.1, 0.3)
```

[70]:
```
True
```

[ ]:
```
x = 0.0
while True:
    x += 0.1
    if x == 0.3:
        break
    else:
        print(x)
```

[71]:
```
x = 0.0
while True:
    x += 0.1
    if math.isclose(x, 0.3):
        break
    else:
        print(x)
```

```
0.1
0.2
```

[ ]:
```
printc = complex(3, 4) + (1+1j)
c
```

명령 줄과 들여쓰기

- 들여쓰기 공백의 규정은 없으나, 공백 4 개 권장 (탭 사용 금지)
- 동일 블록 내에서는 동일 공백을 사용

[72]:
```
fact = False
if fact == True:
    print("Answer")
    print("True")
else:
        print("Answer")
        print("False")
```

```
Answer
False
```

[74]:
```
fact = False
if fact == True:
    print("Answer")
    print("True")
else:
        print("Answer")
    print("False")
```

```
  File "<tokenize>", line 7
    print("False")
    ^
IndentationError: unindent does not match any outer indentation level
```

여러 줄에 나누어 한 명령 줄 작성하기

- Python의 명령줄은 개행 문자(new line)로 끝난다.
- 그러나 Python은 줄 연속 문자(line continuation character) (₩)를 사용하여 현재 문장이 다음 문장과 연결되어 있다는 것을 나타낸다.
- 예를 들면, 다음과 같이 한 문장을 세 줄에 나누어 쓸 수 있다.

[75]:
```
item_1 = 3
item_2 = 4
item_3 = 5
total = item_1 + item_2 + item_3
print(total)
total = item_1 + \
        item_2 + \
        item_3
print(total)
```

```
12
12
```

여러 줄에 나누어 한 명령 줄 작성하기

- [ ], { }, 또는 ( ) 사이에 있는 내용들은 줄 연속 문자가 필요없다.
- 예를 들면, 다음과 같이 한 문장을 두 줄로 나누어 쓸 수 있다.

[ ]:
```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

주석문

- 문자열 외부에 있는 해쉬 문자(#)는 주석의 시작을 의미한다.
- Python 해석기는 어느 줄에서 # 문자 뒤의 모든 내용은 주석으로 간주하여 무시한다.

[76]:
```
# First comment
print("Hello, Python!") # second comment
```

```
Hello, Python!
```

[77]:
```
print("Hello, Python!")
```

```
Hello, Python!
```

- 다음과 같이 여러 줄을 주석으로 처리할 수도 있다.

[93]:
```
# This is a comment.
# This is a comment, too.
# I said that already.
print("Hello, Python!")
```

```
Hello, Python!
```