데이터분석 프로그래밍
연산자

임현기

대입 연산자

- 변수에 값을 저장하는 연산자
- =연산자의 왼쪽은 변수,
  오른쪽은 값, 변수 혹은 수식

```
>>> x = 100 + 200   # x에 100 + 200의 결과를 할당
>>> x
300
```

- 왼쪽에 변수가 아닐 시 문법 오류

```
>>> x = 20
>>> y = 10
>>> 100 = x + y    # 등호의 왼쪽에는 반드시 변수이름이 와야한다
  File "<stdin>", line 1
SyntaxError: cannot assign to literal
```

대입 연산자

- 여러 변수에 동일한 값 저장 가능
- 여러 변수에 여러 값 저장 가능

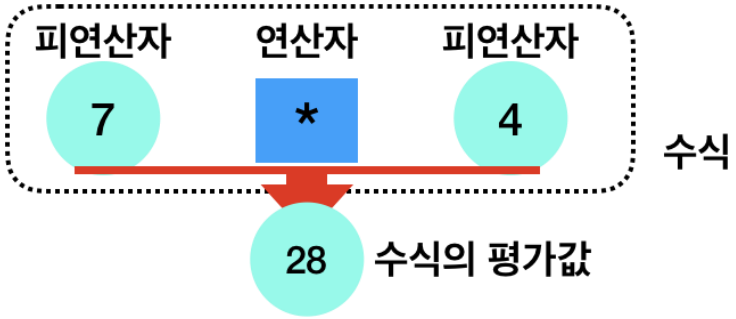
```
>>> x = y = 100            # 여러 변수에 동일한 값을 할당하는 다중 할당문
>>> x, y
(100, 100)
>>> n1, n2 = 100, 200      # 여러 변수에 한꺼번에 여러 값을 할당하는 동시 할당문
>>> n1, n2
(100, 200)
```

잠깐 – 할당 연산자와 변수의 선언

전통적으로 프로그래밍에서 어떤 변수를 사용하려고 하면, 변수의 이름과 자료형을 밝히고 앞으로 이 변수를 사용하겠다는 선언declaration을 해야 한다. 그런데 파이썬은 변수를 선언하지 않고 바로 사용할 수 있다. 우리도 지금까지 변수를 선언한 적이 없다. 하지만 내부적으로는 변수를 선언하는 일이 이루어지는데, 어떤 변수에 할당 연산자가 처음으로 적용되는 순간 우리에게 보이지 않는 방식으로 선언이 이루어진다. 이때 파이썬은 할당 연산자의 오른쪽 수식이 나타내는 값에 따라 변수의 자료형을 결정하게 된다.

수식과 연산자

- 수식이란 피연산자들과 연산자의 조합
  - 연산자: 어떤 연산을 나타내는 기호
  - 피연산자: 연산의 대상이 되는 숫자나 변수



수식과 연산자

- 산술 연산자

연산자	기호	사용예	결과값
덧셈	+	7 + 4	11
뺄셈	–	7 – 4	3
곱셈	*	7 * 4	28
지수	**	7 ** 2	72 = 49
나눗셈(정수 나눗셈의 몫)	//	7 // 4	1
나눗셈(실수 나눗셈)	/	7 / 4	1.75
나머지	%	7 % 4	3

수식과 연산자

- 나눗셈 연산자 /
  - 항상 실수로 계산

```
>>> 7 / 4          # 나눗셈은 항상 실수로 계산된다
1.75
>>> 8 / 4          # 나눗셈이기 때문에 정수 2가 아닌 실수 2.0이 출력
2.0
```

- 몫 연산자 //

```
>>> 7 // 4         # //을 사용해서 나누면 정수 몫만 나온다.
1
>>> 8 // 4         # //을 사용해서 나누면 정수 몫만 나온다.
2
>>> 9 // 4         # //을 사용해서 나누면 정수 몫만 나온다.
2
```

나머지 연산자

- x % y
  - x를 y로 나누어서 남은 나머지 반환
  - ex: 11 % 4는 3

```
p = int(input("분자를 입력하시오: "))
q = int(input("분모를 입력하시오: "))
print("나눗셈의 몫 =", p // q)
print("나눗셈의 나머지 =", p % q)
```

```
분자를 입력하시오: 7
분모를 입력하시오: 4
나눗셈의 몫 = 1
나눗셈의 나머지 = 3
```

거듭제곱 연산자

- 2^7을 파이썬으로 표현하면 2 ** 7

```
>>> 2 ** 7                  # 2의 7제곱이 계산되며 2*2*2*2*2*2*2와 동일함
128
```

- 거듭제곱 연산자는 다른 연산자들보다 높은 우선순위를 가짐
  - 10*2**7 = 10×2^7
- 거듭제곱 연산자는오른쪽에서 왼쪽으로 계산
  - 2**2**3은 2**(2**3)으로 계산 됨

거듭제곱 연산자

- 원금 a, 이자율 r, n년 후 원리금 합계

```
a = 1000
r = 0.05
n = 10
print(a * (1 + r) ** n)
```

```
1628.894626777442
```

거듭제곱 연산자

```
bottom = float(input('직각삼각형의 밑변의 길이를 입력하시오: '))
height = float(input('직각삼각형의 높이를 입력하시오: '))
hypotenuse = (bottom ** 2 + height ** 2) ** 0.5
print('빗변은', hypotenuse, '입니다')
```

```
직각삼각형의 밑변의 길이를 입력하시오: 3
직각삼각형의 높이를 입력하시오: 4
빗변은 5.0 입니다
```

자동 판매기를 시뮬레이션하는 프로그램을 작성해보자. 자동 판매기는 사용자로부터 투입한 돈과 물건값을 입력받는다. 물건값은 100원 단위라고 가정한다. 프로그램은 잔돈을 계산하여 출력한다. 자판기는 동전 500원, 100원짜리만 가지고 있다고 가정하자. 투입한 금액과 물건값을 입력으로 받아 거스름돈으로 500원 동전 몇 개와 100원 동전 몇 개를 내어 보내면 되는지 계산하는 프로그램을 작성하라.

원하는 결과

```
투입한 돈: 5000
물건값: 2600
거스름돈: 2400
500원 동전의 개수: 4
100원 동전의 개수: 4
```

```python
money = int(input("투입한 돈: "))
price = int(input("물건값: "))

change = money - price
print("거스름돈: ", change)
coin500s = change // 500        # 500으로 나누어서 몫이 500원 동전의 개수
change = change % 500           # 500으로 나눈 나머지를 계산한다.
coin100s = change // 100        # 100으로 나누어서 몫이 100원짜리의 개수

print("500원 동전의 개수:", coin500s)
print("100원 동전의 개수:", coin100s)
```

복합 할당 연산자

- 산술 연산자와 할당 연산자를 결합

- 산술 연산자를 @라고 하면

$$a \ @= \ b \iff a \ = \ a \ @ \ b$$

연산자	사용 방법	의미
+=	i += 10	i = i + 10
-=	i -= 10	i = i - 10
*=	i *= 10	i = i * 10
/=	i /= 10	i = i / 10
//=	i //= 10	i = i // 10
%=	i %= 10	i = i % 10
**=	i **= 10	i = i ** 10

비교 연산자

- 두개의 피연산자를 대상으로 크기 관계 연산
- 결과를 True 혹은 False로 반환
  - 부울형

연산자	설명	a = 100 b = 200
==	두 피연산자의 값이 같으면 True를 반환	a == b는 False
!=	두 피연산자의 값이 다르면 True를 반환	a != b는 True
〉	왼쪽 피연산자가 오른쪽 피연산자보다 클 때 True를 반환	a 〉 b는 False
〈	왼쪽 피연산자가 오른쪽 피연산자보다 작을 때 True를 반환	a 〈 b는 True
〉=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 때 True 반환	a 〉= b는 False
〈=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 때 True 반환	a 〈= b는 True

비교 연산자

```
>>> a, b = 100, 200   # a에는 100, b에는 200을 할당하는 동시할당문
>>> a == b
False
>>> a != b
True
>>> a > b
False
>>> a < b
True
>>> a >= b
False
>>> a > = b            # 두 개의 기호로 표현된 비교 연산자는 띄어쓰면 안 된다 : ERROR
File "<stdin>", line 1
a > = b
^
SyntaxError: invalid syntax
>>> a => b             # 두 개 기호로 표현된 비교 연산자의 순서를 뒤집어도 안 된다 : ERROR
File "<stdin>", line 1
a => b
^
SyntaxError: invalid syntax
```

논리 연산자

- 파이썬은 and, or, not 연산자를 지원
- 연산 결과는 True나 False 중 하나로 반환

- 부울 자료형 True, False 대신 자료형의 데이터도 부울형으로 변환 가능
  - 0은 False, 0을 제외한 모든 수는 True
  - ''와 같은 공백문자는 False, 그 외 문자는 True
  - None은 값이 없음을 의미 → False

논리 연산자

```
>>> 10 > 20         # 10은 20보다 작으므로 10 > 20은 False가 됨
False
>>> 10 < 20         # 10은 20보다 작으므로 10 < 20은 True가 됨
True
>>> bool(9)         # 9는 0이 아니므로 True가 됨
True
>>> bool(-1)        # -1 역시 0이 아니므로 True가 됨
True
>>> bool(0)         # 숫자 값중에서는 유일하게 0의 값만 False가 됨
False
>>> bool(None)      # None은 값이 없음을 표현함, 따라서 False가 됨
False
>>> bool('')        # 빈 문자열이므로 False가 됨
False
>>> bool('hello')   # 문자열 값이 있으므로 True가 됨
True
```

논리 연산자

연산자	의미
x **and** y	x와 y중 거짓(False)이 하나라도 있으면 거짓이 되며 모두 참(True)인 경우에만 참이다.
x **or** y	x나 y중에서 하나라도 참이면 참이 되며, 모두 거짓일 때만 거짓이 된다.
not x	x가 참이면 거짓, x가 거짓이면 참이 된다.

x and *y*

x	*y*	*x* and *y*
False	False	False
False	True	False
True	False	False
True	True	True

x or *y*

x	*y*	*x* or *y*
False	False	False
False	True	True
True	False	True
True	True	True

not *x*

x	not *x*
False	True
True	False

비트 연산자

- 정수 데이터형에 대해 비트 단위의 조작이 가능
  - 비트 연산자

연산자	의미	설명
&	비트 단위 AND	두 개의 피연산자의 해당 비트가 모두 1이면 1, 아니면 0
|	비트 단위 OR	두 피연산자의 해당 비트 중 하나라도 1이면 1, 아니면 0
^	비트 단위 XOR	두 개의 피연산자의 해당 비트의 값이 같으면 0, 아니면 1
~	비트 단위 NOT	0은 1로 만들고, 1은 0으로 만든다.
<<	비트 단위 왼쪽으로 이동	지정된 개수만큼 모든 비트를 왼쪽으로 이동시킨다.
>>	비트 단위 오른쪽으로 이동	지정된 개수만큼 모든 비트를 오른쪽을 이동시킨다.

비트 연산자

- 연산자 &는 두 개의 피연산자가 가진 비트들을 하나씩 짝을 지어 각각에 대해 연산을 적용
- 짝이 되는 비트가 모두 1이면 1,
  아니면 0을 반환
- 1을 True, 0을 False로 생각할 때 이 연산자는 각 비트에 대해 and 연산자와 비슷한 일을 수행
- 연산자 |는 비트 단위로 OR 연산을 수행,
  연산자 ~는 비트 단위로 NOT 연산을 수행

비트 연산자

- 어떤 정수의 이진수 값을 확인하고 싶으면 bin() 함수를 사용 가능

```
>>> bin(9)          # 2진수 형식 출력 (00001001)
'0b1001'
>>> bin(10)         # 2진수 형식 출력 (00001010)
'0b1010'
>>> bin(9 & 10)     # 9와 10을 2진수로 표현했을 때 모두 1인 위치만 1 나머지는 0이 된다
'0b1000'
>>> bin(9 | 10)     # 9와 10을 2진수로 표현했을 때 하나로도 1이 나타나는 자리는 1이 된다
'0b1011'
>>> bin(~9)         # 9의 모든 비트를 바꾸면 음수 10이 된다. 이것은 아래에 설명한다
'-0b1010'
```

비트 연산자

- ^ 연산자 XOR 연산 수행
  - 해당 비트의 값이 같으면 0, 아니면 1을 반환

```
>>> bin(9)       # 이진수 00001001
'0b1001'
>>> bin(10)      # 이진수 00001010
'0b1010'
>>> 9 ^ 10       # 결과는 00000011 = 십진수 3
3
>>> bin(9 ^ 10) # 앞 부분의 0은 모두 사라지고 1이 나타나는 곳부터 출력된다.
'0b11'
```

비트 연산자

- 쉬프트 연산
  - << 연산자는 지정된 수만큼 모든 비트를 왼쪽을 이동
  - >> 연산자는 반대로 지정된 수만큼 모든 비트를 오른쪽으로 이동
  - 비트가 한칸 왼쪽으로 이동하면 정수는 2배의 값을 가지게 되고, 한 칸 오른쪽으로 이동하면 //2 연산을 한 결과가 됨

비트 연산자

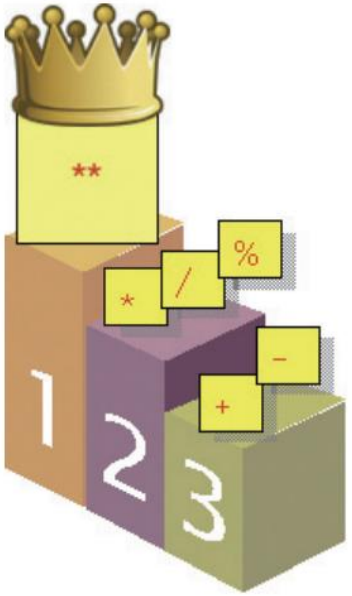
연산자	사용 예시	의미
&=	i &= 10	i = i & 10
|=	i |= 10	i = i | 10
^=	i ^= 10	i = i ^ 10
<<=	i <<= 10	i = i << 10
>>=	i >>= 10	i = i >> 10

```
>>> num = 2      # 정수 2는 이진수 10(2)으로 표현됨
>>> num
2
>>> num <<= 1    # 정수 2를 1비트 왼쪽 이동시키면 100(4)가 됨
>>> num
4
>>> num <<= 1    # 정수 4를 1비트 더 왼쪽 이동시키면 1000(8)이 됨
>>> num
8
>>> num >>= 1    # 정수 8를 1비트 오른쪽쪽 이동시키면 100(4)가 됨
>>> num
4
```

연산자 우선순위

$x + y * z$

① ②



$x + y * z$

① ②

$(x + y) * z$

① ②

연산자 우선순위

연산자	설명
**	지수 연산자
~ , + , -	단항 연산자
* , / , % , //	곱셈, 나눗셈, 나머지 연산자
+ , -	덧셈, 뺄셈
>> , <<	비트 이동 연산자
&	비트 AND 연산자
^ , |	비트 XOR 연산자, 비트 OR 연산자
<= , < , > , >=	비교 연산자
== , !=	동등 연산자
= , %= , /= , //= , -= , += , *= , **=	할당 연산자, 복합 할당 연산자
is , is not	아이덴티티 연산자
in , not in	소속 연산자
not , or , and	논리 연산자

높은
우선순위

random 모듈과 math 모듈

- random 모듈

```
>>> import random
>>> random.random()            # 0 이상 1 미만의 임의의 실수를 반환
0.19452357419514088
>>> random.random()            # 이 함수는 매번 다른 실수를 반환
0.6947454047320903
>>> random.randint(1, 7)       # 1 이상 7 이하(7을 포함)의 임의의 정수를 반환
3
>>> random.randrange(7)        # 0 이상 7 미만(7을 포함하지 않음)의 임의의 정수를 반환
3
>>> random.randrange(1, 7)     # 1 이상 7 미만(7을 포함안함)의 임의의 정수를 반환
6
>>> random.randrange(0, 10, 2) # 0, 2, 4, 8 중(10은 포함 안함) 하나를 반환함
2
>>> lst = [10, 20, 30, 40, 50] # 여러개의 값을 가지는 리스트를 생성함
>>> random.shuffle(lst)        # lst 리스트의 순서를 무작위로 섞음
>>> lst
[40, 50, 10, 20, 30]
>>> random.choice(lst)         # lst 리스트의 원소 중 무작위로 하나를 고름
30
```

random 모듈과 math 모듈

- math 모듈

```
>>> import math
>>> math.pow(3, 3)       # 3의 3 제곱
27.0
>>> math.fabs(-99)       # -99의 실수 절대값
99.0
>>> math.log(2.71828)
0.999999327347282
>>> math.log(100, 10)    # 로그 10을 밑으로 하는 100값
2
>>> math.pi              # 원주율
3.141592653589793
>>> math.sin(math.pi / 2.0)      # sin() 함수의 인자로 PI/2.0를 넣어보자
1.0
```