

산술 연산자

★ 수식(expression) : 피연산자들과 연산자의 조합

★ 산술 연산자 표

(*) 양수의 경우 부호 생략 가능

operator	meaning	example
+	덧셈 또는 + 부호	+10(*) 15 + 20
-	뺄셈 또는 - 부호	-10 2.5 - 1.5
*	곱셈	10 * 20 = 200
/	나눗셈(실수)	1 / 2 = 0.5 (결과는 항상 실수)
%	나눈 후 나머지 (modulo)	9 % 4 = 1 (9를 4로 나눈 나머지)
//	나눈 후 몫 (floor division)	15 // 4 = 3 (15를 4로 나눈 몫)
**	지수승	2**4 = 16 ($2^4 = 16$)

★ 피연산자가 모두 정수형인 경우는 결과도 정수형이지만, 하나라도 아닌 경우의 결과는 실수형임 (/ 연산자의 결과는 항상 실수)

```
a = 10; b = 4; x = 10.0; y = 4.0
```

```
print(a + b)        # a + b = 10 + 4 = 14
print(a + x)        # a + x = 10 + 10.0 = 20.0
print(a * y)        # a * y = 10 * 4.0 = 40.0
```

산술 연산자

- ★ 나머지(modulo) 연산자는 어떤 숫자의 홀/짝수 여부, 임의의 수의 배수인지 판단할 때 많이 사용됨

```
a = 5
```

```
b = 30
```

```
print(a % 2)    # 1    5는 홀수
```

```
print(b % 3)    # 0    30은 3의 배수
```

산술 연산자

★ Multi-line statement

- 하나의 statement를 여러 줄을 사용하여 작성해야 할 때는 backslash(₩로 표시)를 사용

```
>>> a = 1 + 2 + 3 + ₩  
      4 + 5 + 6 + ₩  
      7 + 8 + 9
```

명령어 입력 후 엔터키를 치면 입력된 명령어를 실행하지만, 이와 같이 ₩를 입력하고 엔터키를 치면 여러줄이 필요한 명령어를 계속 입력할 수 있음

- 특정한 data type을 표시할 때 사용되는 (...), [...], {...} 등의 괄호 내부에서는 backslash없이 줄을 바꾸어도 무방

★ 여러 수식의 계산 결과 출력

- 콤마로 구분하여 여러 수식을 입력하면, 결과는 튜플 데이터 형으로 출력 (추후에 자세히 다시 설명)
- print() 함수의 인수로 수식을 주면 계산한 결과를 출력

```
>>> 6+4, 6-4, 6*4, 6/4 #괄호가 생략되어 있음 (6+4, 6-4, 6*4, 6/4)  
(10, 2, 24, 1.5)  
>>> print(6+4, 6-4, 6*4, 6/4)  
10 2 24 1.5
```

복합 연산자

★ += 처럼 대입 연산자와 다른 연산자를 합쳐 놓은 연산자

assingment	example	description
+=	$x += y$	$x = x + y$ 와 동일
-=	$x -= y$	$x = x - y$ 와 동일
*=	$x *= y$	$x = x * y$ 와 동일
/=	$x /= y$	$x = x / y$ 와 동일
//=	$x //= y$	$x = x // y$ 와 동일
%=	$x \% = y$	$x = x \% y$ 와 동일
**=	$x ** = y$	$x = x ** y$ 와 동일

관계 연산자(Relational Operators)

★ 관계 연산자

- 두 값을 비교하는 연산자 : 결과는 True(참) 또는 False(거짓)

operator	description	example
==	equal	5 == 7 # False
!=	not equal	5 != 7 # True
>	greater than	5 > 7 # False
<	less than	5 < 7 # True
>=	greater than or equal	5 >= 7 # False
<=	less than or equal	5 <= 7 # True

```
>>> a= 100; b=100;
```

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

```
True
```

두 수가 같으면 True

```
>>> print(a!=b)
```

```
False
```

```
>>> print(a>b, a<b)
```

```
False False
```

```
>>> print(a<=b, a>=b)
```

```
True True
```

논리 연산자

★ 논리 연산자의 종류

- 복잡한 조건을 표현하려면 논리연산자를 사용
- 몇 개의 조건식을 조합하여 명령문의 수행여부를 결정할 때 사용

operator	description	example
and	logical and. ~이고 그리고	모두 True이어야 True
or	logical or. ~이거나 또는	하나라도 True이면 True
not	negates the truth value. 부정	참이면 거짓. 거짓이면 참

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

논리 연산자

★ 해당 년도가 윤년인지 확인하기

- 윤년의 정의

: 4로 나눠 떨어져야 하고, 100으로 나눠 떨어지면 안 됨. 또는 400으로 나눠 떨어지면 윤년

```
year = 2020
```

```
if ((year % 4 == 0) and (year % 100 != 0)) or (year % 400 == 0):
```

```
    print("윤년")
```


```
else :
```

```
    print("윤년 아님")
```

해당 년도가 윤년의 조건을 만족하면 "윤년"을 출력하고, 아니면 "윤년 아님"을 출력

연산자 우선순위

우선순위

()	anything in brackets is done first	Highest
**	exponentiation	
-x, +x	arithmetic operators	
*, /, %, //	arithmetic operators	
+, -	arithmetic operators	
<, >, <=, >=, !=, ==	relational operators	
=, +=, -=, *=, etc	assignment operators	
not	logical operator	
and	logical operator	Lowest
or	logical operator	

- ★ 같은 우선순위를 갖는 operator는 왼쪽부터 계산
- ★ 단, ** operator는 오른쪽부터 (예: $2^{**}2^{**}3 = 2^{**}8 = 256$)
- ★ 애매하면 괄호 () 를 사용 (예: $(2^{**}2)^{**}3 = 4^{**}3 = 64$)

내장 함수

★ 수치연산 관련 내장 함수 (built-in function)

```
>>> abs(-10)           # 절대값 반환
10
>>> 0.1 + 0.2           # 실수는 근사치 사용, 정확한 0.3 이 아님
0.30000000000000004
>>> 0.1 + 0.1 + 0.1 == 0.3
False
>>> round(0.1 + 0.1 + 0.1, 10) == round(0.3, 10)
True
>>> round(3.123456, 4) # 반올림 후 소수점 이하 자릿수 4
3.1235
```

round(실수) 또는 round(실수, 자릿수)

: 자릿수는 반올림 후의 소수점 이하 자릿수를 의미하며, 지정하지 않으면 정수 반환

math Module

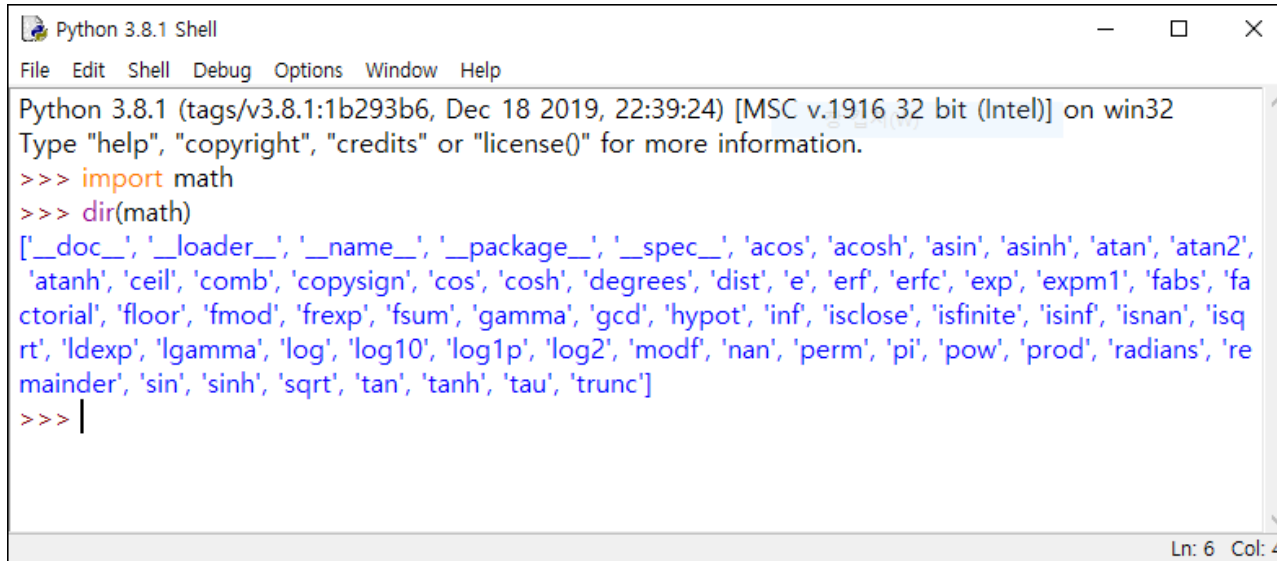
- ★ python의 math 관련 함수들을 모아둔 모듈
- ★ math 모듈의 함수를 사용하기 위한 import 문(3가지 방법)

```
from math import *      # 이 경우 함수 사용시, 모듈 이름이 불필요
a = sqrt(4.0)           # sqrt() 함수를 함수명으로만 호출
print(a)                # 2.0 출력
```

```
import math             # 이 경우 math.을 붙여야 함
a = math.trunc(1.5)     # trunc() 함수 앞에 해당 모듈명을 명시해야 함
print(a)               # 1 출력
```

```
import math as m        # 이 경우 m.을 붙여야 함
a = m.pow(81, 0.5)      # m은 math의 별칭에 해당
print(a)               # 9.0 출력
```

math Module



```
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import math
>>> dir(math)
['_doc_', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'fa
ctorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isq
rt', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'perm', 'pi', 'pow', 'prod', 'radians', 're
mainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>> |
```

Ln: 6 Col: 4

```
>>> import math
>>> num = 81
>>> print(num**0.5, math.pow(num, 0.5), math.sqrt(num))
9.0 9.0 9.0
>>>
```