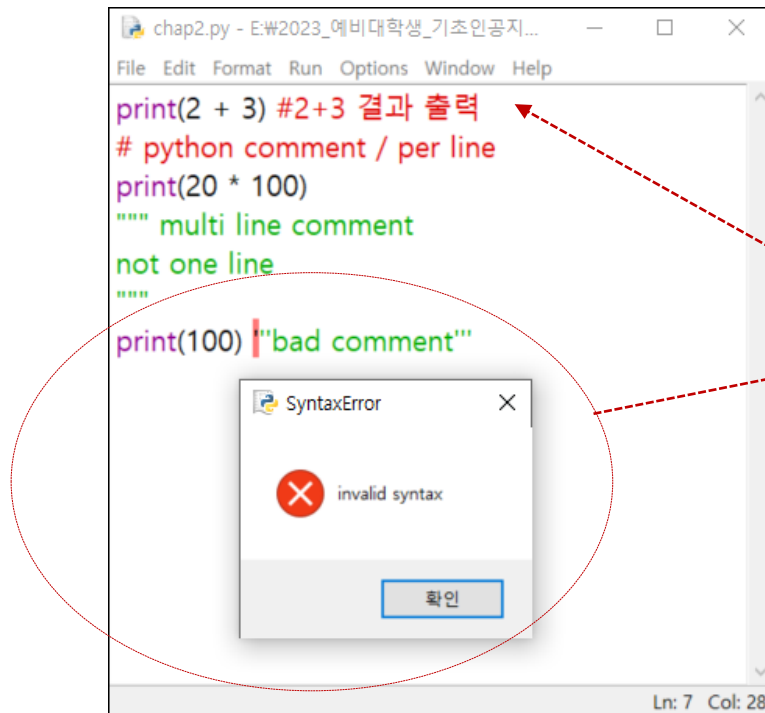


기초 인공지능 프로그래밍

2장 변수와 데이터 형식

주석 (comment)

- 주석은 프로그램 설명문과 같음(명령어가 아님)
- 파이썬 인터프리터(interpreter)는 명령어 실행 때 주석은 무시
- # 문자는 한 줄짜리 주석에 사용
- 여러 줄 주석은 큰 따옴표 또는 작은 따옴표 세 개를 사용
 - `"""` (`'''`)로 시작하면 `"""` (`'''`)로 마쳐야 함



마지막 명령어에서 error 발생
(명령어 뒤에 주석을 추가할 때는
첫번째 명령어와 같이 # 문자로
주석 처리 해야함)

변수

- 프로그램 실행 시 처리되는 모든 종류의 데이터들은 메모리에 존재
 - 또한 파이썬 프로그램의 모든 데이터(값)는 객체이며 메모리에 존재
- 변수는 메모리에 저장된 값(value)을 참조하는 이름
 - 즉, 변수는 메모리에 생성된 객체(object)을 참조하는 수단
- 파이썬 변수
 - 변수는 값을 처음 할당 받는 순간에 생성(따로 변수 선언이 필요 없음)
 - 변수가 참조하는 값은 실행 중간에 언제든지 변경 가능

```
x = 10    # 변수 x가 생성되며, 정수 10을 값으로 하는 정수 객체를 참조
y = 10    # 변수 y가 생성되며, 변수 x와 같은 객체를 참조
x = 2.5    # 변수 x는 실수 2.5를 값으로 하는 실수 객체를 참조하도록 변경
```

변수명 규칙

- 변수명은 영어 대소문자와 숫자, 밑줄(_)만 허용
 - 영어 소문자와 대문자를 구분(다른 변수명으로 간주)
 - 변수명에 공백을 허용하지 않으므로 단어를 구분하려면 밑줄 (_)을 사용하거나, myNewCar와 같이 낙타체 표기법(대소문자로 단어 구별)을 사용하여 의미를 부여할 수 있음
 - 변수명은 의미 있는 이름을 사용하는 것이 효율적임

	변수명 규칙
1	변수명은 영어 대소문자, 숫자, 밑줄(_)로만 이루어짐 다른 기호를 사용하면 구문 에러(Syntax Error) (예) Money\$, My Score : 문자 \$와 공백은 사용하여 구문 에러
2	변수명은 영어 대소문자 또는 밑줄로만 시작, 숫자로 시작 하면 안됨 (예) 7up, 5brothers : 숫자로 시작했기 때문에 구문 에러
3	파이썬 지정단어(Keyword, Reserved word)들은 변수명으로 사용할 수 없음
4	파이썬에서는 대문자와 소문자를 구분 (hour 와 Hour는 다른 변수)

파이썬 지정단어
(Keyword/Reserve
Word)

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

변수 할당문

- 변수는 대입문/할당문(assign 명령어)에 의해 생성

variable = 값

 **할당 연산자(대입 연산자)**

- 할당 연산자의 의미는 오른쪽 명령어의 실행 결과 값을 왼쪽 변수에게 할당하는 것
- 따라서 할당 연산자의 **왼쪽에는 변수만** 올 수 있고, 오른쪽에는 무엇이든 (값, 변수, 수식, 함수 등) 올 수 있음
- 파이썬의 연산자 **=** 는 수학에서의 equality(==)와 혼동하면 안됨
- 변수 초기화(initialization)란 변수에 처음 값을 할당하는 것을 의미함
- 할당 연산자의 오른쪽에 사용되는 변수는 반드시 그 전에 값이 할당된 변수이어야 함

```
n1 = 10 * 20
n2 = n1
print(n1, n2)
n1 = n3
print(n1, n3)
```

python script code

```
>>>
===== RESTART: C:\Users\Wadmin\Desktop\script\Wtest.py =
200 200
Traceback (most recent call last):
  File "C:\Users\Wadmin\Desktop\script\Wtest.py", line 4, in <module>
    n1 = n3
NameError: name 'n3' is not defined
>>> |
```

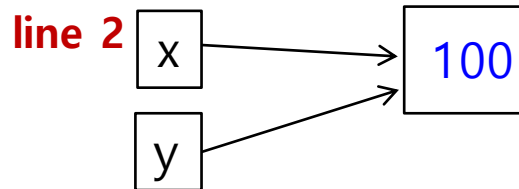
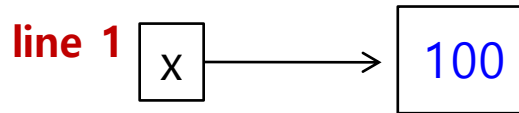
변수 사용

- 이미 생성된 변수에 다른 형태의 데이터(자료형) 값도 할당 할 수 있음
 - 즉, 동일한 변수에 필요에 따라 어떤 자료형의 데이터도 저장할 수 있음

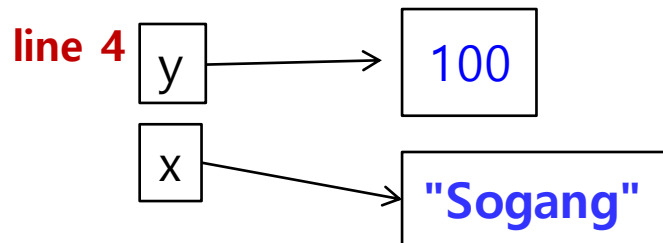
python script code

```
x = 100  
y = x  
x = "Sogang"
```

`y = x` 명령어는 `x`가 참조하는 객체(object)를 같이 참조하는 변수 `y`를 생성하는 것을 의미함



변수 `x`는 정수 형태의 데이터를 참조하다가, 문자들로 이루어진 데이터를 할당 받아 참조



자료형(data types)

- 변수는 정수, 실수, 문자열 등의 다양한 자료형의 데이터를 할당 받아 저장함
- 파이썬에서 제공하는 기본 자료형
 - 수치 자료형(numbers)
 - 정수(integer, int) : 7, 123, -256,...
 - 실수(float) : 3.14, -1.2345, -3.0e5,...
 - 복소수(complex) : 2.5+3.2j, 1+2j, 3+1j,... (본 강의에서는 다루지 않음)
 - 부울 자료형(Boolean) : True, False
 - 문자열(string, str) : "Hello", 'Hello',...
 - 리스트(list) : [1, 2.2, "Hello"]
 - 튜플(tuple) : (5.5, -3, "Hello")
 - 집합(set) : {1, 2, 3, 4, 5}
 - 사전(dictionary, dict) : {'val1':1, 'val2':2}

차후 자세히 설명

자료형(data types)

- 정수 (int)
 - 정수는 소수점이 없는 숫자 데이터(100, -123, 0, ...)
 - int는 기본 정수 자료형
 - 파이썬 버전 3부터는 정수의 크기에 제한이 없음(이론적으로)
- 정수 데이터 표현
 - 16진수는 0x나 0X (숫자 0)
 - 8진수는 0o나 0O(숫자 0 + 알파벳 o(O))
 - 2진수는 0b나 0B를 접두사로 붙여 표현

```
>>> print(0xFF, 0o77, 0b1111)
255 63 15
>>>
```


자료형(data types)

- 여러 진법으로 정수 표현

10진수	2진수	16진수
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

자료형(data types)

- 여러 진법으로 정수 표현

```
>>> hex(0)
'0x0'
>>> hex(255)
'0xff'
>>> a = 0xFF
>>> a
255
>>> b = 0x20
>>> b
32
>>> bin(0)
'0b0'
>>> bin(8)
'0b1000'
>>> a = 0b1001
>>> a
9
>>> b = 0b11111111
>>> b
255
```

```
>>> oct(8)
'0o10'
>>> oct(10)
'0o12'
>>> oct(64)
'0o100'
>>> a = 0o10
>>> a
8
>>> b = 0o12
>>> b
10
>>> c = 0o100
>>> c
64
```

hex(), bin(), oct()
함수들은 인자로
받은 10진수
정수를 해당
진수로 변환,
문자열 형태로
반환

자료형(data types)

- 실수 (float)

- 실수는 소수점이 있는 숫자 데이터(3.14, -2.7, ...)
- 실수의 두가지 표현 방식

고정 소수점 방식: 132.234, -0.023 등과 같이 소수점을 고정하여 표시한 방법

부동 소수점 방식: 1.531e+35, 3.54e-64 등과 같이 지수형식 (exponential form)으로 표시하는 방법

- 실수를 표현할 때 오차가 발생할 수 있음

```
>>> print(0.1 + 0.2)
0.30000000000000004
>>> print(4.3 - 2.7)
1.5999999999999996
>>>
```

정확한 0.3 또는
1.6이 아님

```
>>> print(3.1e+8)
310000000.0
>>> print(3.1e-3)
0.0031
>>>
```

자료형(data types)

- 부울 (boolean)
 - True, False 값을 저장(참, 거짓을 의미)
 - 부울은 단독으로 사용하기 보다 if 조건문, 반복문 등 함께 주로 사용
- 문자열(str)
 - 문자열은 양쪽을 큰따옴표(")나 작은따옴표(')로 감싼 문자들의 모임

```
>>> print(100 + 200)
```

```
300
```

```
>>> print("100" + "200")
```

```
100200
```

```
>>>
```

→ 100+200 은 정수+정수 연산
결과를 출력

→ "100"+"200"은 문자열과 문자열을 결합한
새로운 문자열을 생성하는 연산 결과를 출력

자료형(data types)

- 내장 함수 type()
 - 객체의 자료형을 알려주는 함수
 - 프로그램 실행 중, 변수(variable)가 어떤 자료형을 참조하는지 확인할 수 있음

```
>>> x = 10
>>> print(type(x)) 변수 x가 참조하는 객체의 자료형 출력
<class 'int'> 정수
>>> x = 2.5
>>> print(type(x))
<class 'float'> 실수
>>> x = "Sogang"
>>> print(type(x))
<class 'str'> 문자열
>>>
```

자료형 변환

- 정수, 실수, 문자열 등의 자료형들을 다른 자료형으로 변환할 수 있으며, 파이썬은 이를 위한 내장 함수를 제공함
 - int() 함수**
 - 숫자로만 구성된(숫자를 제외한 다른 문자가 없는) 문자열을 정수로 변환
 - 소수점이 있는 숫자 형태(실수 형태)의 문자열은 정수로 변환할 수 없음 : **error 발생**
 - 실수 값은 정수로 변환
 - float() 함수**
 - 실수 형태 또는 정수 형태의 문자열을 실수로 변환
 - 정수 값은 실수로 변환
 - str() 함수**
 - 정수나 실수 값을 문자열로 변환

python
script
code

```
a = int("34") + int(3.8)
print(a)
b = "Python" + str(6) + "class"
print(b)
```

실행 결과

```
===== RESTA
37
Python6class
>>>
```

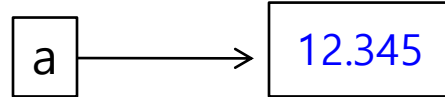
```
>>> a = int("34.5")
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    a = int("34.5")
ValueError: invalid literal for int() with base 10: '34.5'
>>> |
```

자료형 변환 예시

python script code

```
a = 12.345
```

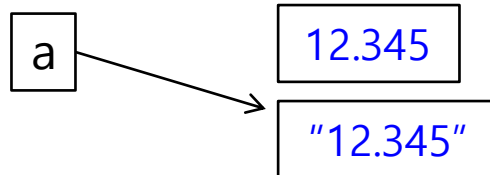
(1)



변수 a에 실수 12.345를 할당

```
a = str(a)  
print(a, type(a))
```

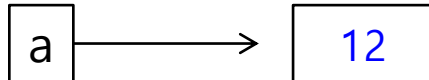
(2)



변수 a가 변환된 문자열 데이터를 가리키게 함

```
a = int(float(a))  
print(a, type(a))
```

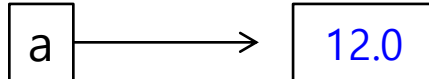
(3)



문자열 데이터에 소수점이 있는 숫자 형태의 데이터는 정수로 변환할 수 없으므로 먼저 float로 변환한 후에 int로 변환

```
a = float(a)  
print(a, type(a))
```

(4)



int 데이터를 float로 변환

```
>>> a = oct(8)  
>>> a  
'0o10'  
>>> b = int(a)  
Traceback (most recent call last):  
  File "<pyshell#2>", line 1, in <module>  
    b = int(a)  
ValueError: invalid literal for int() with base 10: '0o10'  
>>> b = int(a, 8)  
>>> b  
8
```

int() 함수의 두번째 인자의 의미는 첫번째 인자의 진수를 명시하는 것임. 없으면 디폴트로 10진수

```
>>> h1 = hex(97)  
>>> h2 = hex(98)  
>>> r1 = h1+h2  
>>> print(r1)  
0x610x62
```

```
>>> i1 = int(h1, 16)  
>>> i2 = int(h2, 16)  
>>> r2 = i1+i2  
>>> print(r2)  
195  
>>> print(hex(r2))  
0xc3  
>>>
```

여러 변수 할당문(multiple assignments)

- 다수의 변수들이 동시에 하나의 객체를 참조할 수 있음
- 또한, 여러 변수에 서로 다른 값들을 동시에 할당도 가능(동시 할당문)

python script
code

```
x = y = z = 10
```

```
print(x, y, z)
```

```
a, b, c = 5, 4.1, "Hi!"
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

같은 기능의 명령어

```
a = 5; b = 4.1; c = "Hi!"
```

한 줄에 두 개 이상의 명령어를 입력할 때는
semicolon(;)으로 구분해야 함

```
10 10 10
```

```
5
```

```
4.1
```

```
Hi!
```

```
>>>
```

- 변수 num1과 num2의 값을 서로 바꾸는 코드

python script
code

```
num1 = 10; num2 = 20
```

```
print(num1, num2)
```

```
tmp = num1
```

```
num1 = num2
```

```
num2 = tmp
```

```
print(num1, num2)
```

같은 기능의 명령어

```
num1, num2 = num2, num1
```

```
10 20
```

```
20 10
```

```
>>>
```


산술 연산자

■ 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)

- 피연산자들이 모두 정수형인 경우는 결과도 정수형이지만, 하나라도 아닌 경우의 결과는 실수형임 (예외로 / 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 숫자 데이터의 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
```

	4	← // 연산자 결과
7	30	
	28	
	2	← % 연산자 결과

- 할당 연산자와 산술 연산자

```
a = 10; b = 20
c = a + b      # a와 b 값을 더해서 c에 저장
a = a + 100    # a 값을 100 증가
b = b + a      # b 값을 a 값 만큼 증가
print(a, b, c) # 110 130 30 출력
```

산술 연산자

- Multi-line statement

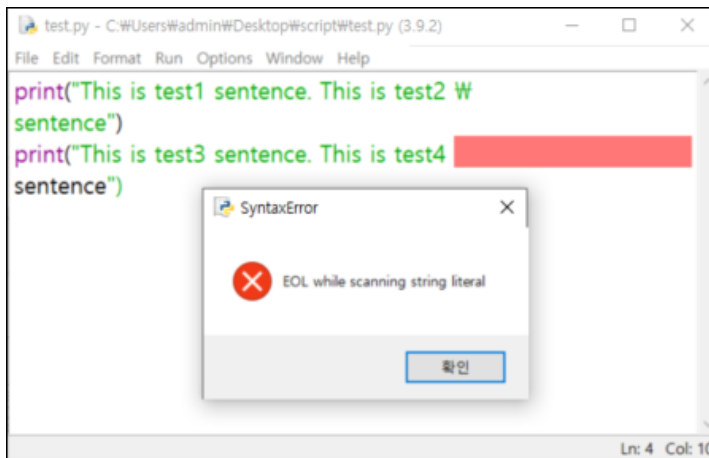
- 하나의 statement를 여러 줄을 사용하여 작성해야 할 때는 backslash(₩로 표시)를 사용

```
a = 1 + 2 + 3 + 4 + ₩  
    5 + 6  
print(a)  
b = 10 + 20 + 30 + 40  
    + 50 + 60  
print(b)
```

명령어 입력 후 엔터키를 치면 명령어 입력이 완료 되었다고 간주하지만, 이와 같이 ₩를 입력하고 엔터키를 치면 여러 줄이 필요한 명령어를 계속 입력할 수 있음

```
21  
100  
>>>
```

10부터 60까지의 수를 더한 값을 구하지 않고 엔터키 입력 전 40까지만 명령어로 간주하여 더함



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

복합 연산자

- += 형태와 같이 대입 연산자와 다른 연산자를 합쳐 놓은 연산자

	example	description
<code>+=</code>	<code>x += y</code>	<code>x = x + y</code> 와 동일
<code>-=</code>	<code>x -= y</code>	<code>x = x - y</code> 와 동일
<code>*=</code>	<code>x *= y</code>	<code>x = x * y</code> 와 동일
<code>/=</code>	<code>x /= y</code>	<code>x = x / y</code> 와 동일
<code>//=</code>	<code>x //= y</code>	<code>x = x // y</code> 와 동일
<code>%=</code>	<code>x %= y</code>	<code>x = x % y</code> 와 동일
<code>**=</code>	<code>x **= y</code>	<code>x = x ** y</code> 와 동일

관계 연산자 (relational operator)

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

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

```
>>> a = 100; b = a
>>> print(a >= b)
True
>>> print(a > b, a < b)
False False
>>>
```

논리 연산자 (logical operator)

- 복잡한 조건을 표현할 때 논리 연산자 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

윤년의 정의

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

```
>>> a = 99
>>> print( (a >= 50) and (a <= 100))
True
>>> print(50 <= a <= 100)
True
>>> print(not(a == 100))
True
>>>
```


```
[25] 1 year = 2022
      2 if ((year % 4 == 0) and (year % 100 != 0)) or (year % 400 == 0):
      3     print("Leap year")
      4 else:
      5     print("No Leap year")
```

No Leap year

해당 년도가 윤년의 조건을 만족
하면 "Leap year"을 출력하고, 아
니면 "No Leap year"을 출력

연산자 우선 순위

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	
or	logical operator	Lowest

- 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

- 수치 연산 관련 내장 함수 (built-in function)

```
n1 = abs(-10)
```

abs() : 절대값 반환 함수

```
n2 = abs(-34.5)
```

```
print(n1, n2)
```

```
n3 = 0.1+0.1+0.1
```

```
print(n3)
```

```
print(n3 == 0.3)
```

```
n3 = round(n3, 10)
```

```
print(n3)
```

```
print(n3 == 0.3)
```

```
n4 = round(3.1234567, 4)
```

```
print(n4)
```

round(실수) 또는 round(실수, 자릿수)
: 자릿수는 반올림 후의 소수점 이하 자릿수를 의미하며,
지정하지 않으면 정수 반환

```
10 34.5
```

```
0.30000000000000004
```

```
False
```

```
0.3
```

```
True
```

```
3.1235
```

```
>>>
```


math Module

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

```
from math import *
```

```
a = sqrt(4.0)  
print(a)
```

이 경우 함수 사용시, 모듈 이름이 불필요
sqrt() 함수를 함수명으로만 호출
2.0 출력

```
import math
```

```
a = math.trunc(1.5)  
print(a)
```

이 경우 math.을 붙여야 함
trunc() 함수 앞에 해당 모듈명을 명시해야 함
1 출력

```
import math as m
```

```
a = m.pow(81, 0.5)  
print(a)  
print(m.floor(4.7))  
print(m.ceil(4.7))
```

이 경우 m.을 붙여야 함
m은 math의 별칭에 해당
9.0 출력
4 출력 : 4.7 이하의 정수 중, 가장 큰 정수
5 출력 : 4.7 이상의 정수 중, 가장 작은 정수

```
>>> import math  
>>> dir(math)
```

math 모듈의 함수 목록을 확인할 수 있음.
(sqrt(), trunc(), pow(), floor(), ceil() 함수는 math 모듈의 함수)