

# Python을 활용한 데이터 분석 강의

연산자, 변수, 자료형

# 산술연산자

산술연산자	사용법	뜻
+	1 + 1	덧셈
-	2 - 2	뺄셈
*	3 * 3	곱셈
/	4 / 4	나눗셈(몫)
%	5 % 3	나눗셈(나머지)
**	6 ** 2	거듭제곱

사칙연산 시 괄호 먼저, 곱셈/나눗셈 먼저

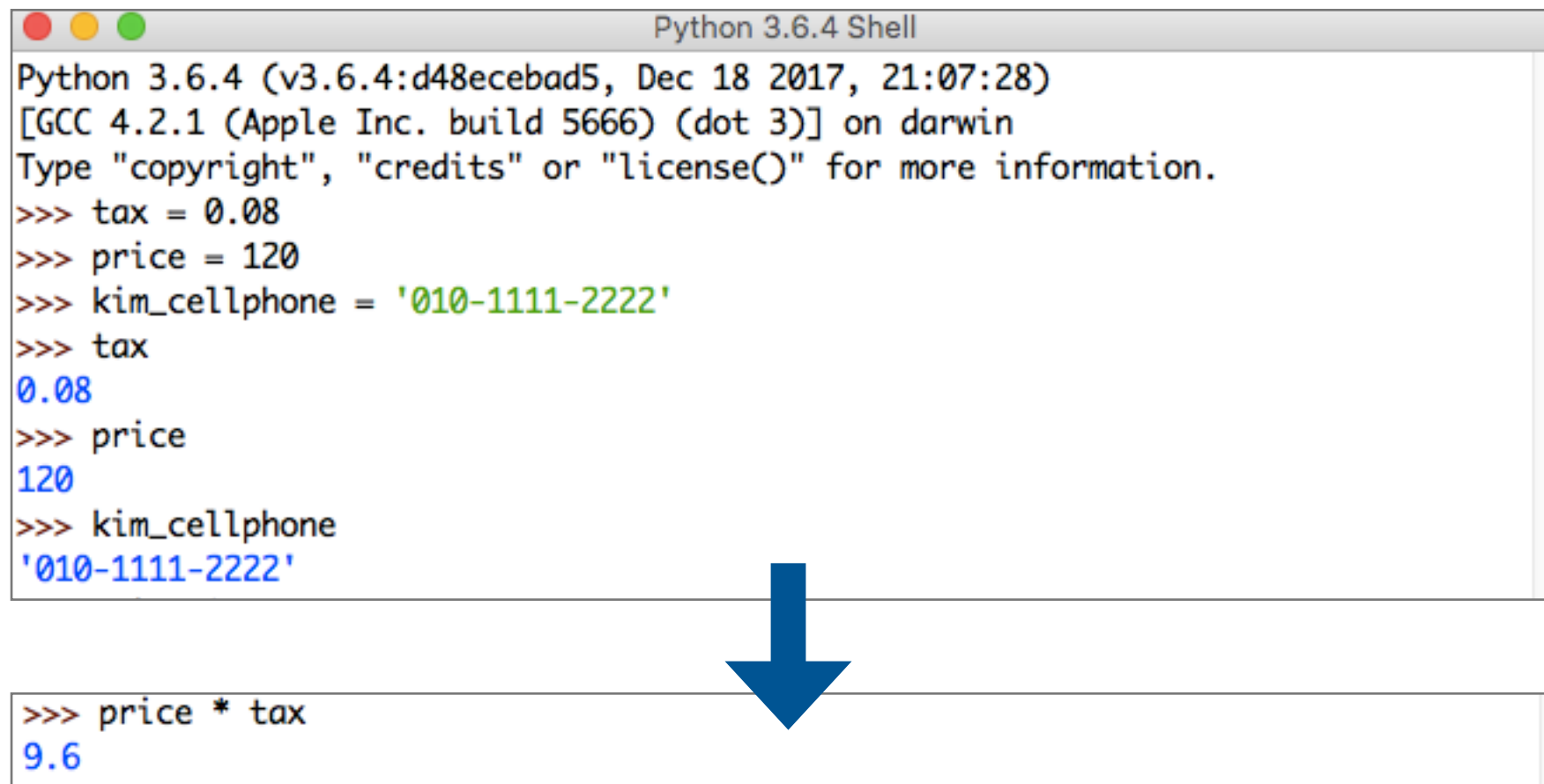
# 비교연산자

비교연산자	예	뜻
>	$x > y$	x는 y보다 크다
>=	$x >= y$	x는 y와 같거나 크다
<	$x < y$	x는 y보다 작다
<=	$x <= y$	x는 y와 같거나 작다
==	$x == y$	x와 y는 같다
!=	$x != y$	x와 y는 같지 않다

'=' != '=='

# 변수

- “변수 = 값”



A screenshot of a Python 3.6.4 Shell window. The window title is "Python 3.6.4 Shell". The text inside shows the Python version and build information, followed by several lines of code and their outputs. A large blue arrow points from the first block of code to the second block.

```
Python 3.6.4 (v3.6.4:d48ecebad5, Dec 18 2017, 21:07:28)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> tax = 0.08
>>> price = 120
>>> kim_cellphone = '010-1111-2222'
>>> tax
0.08
>>> price
120
>>> kim_cellphone
'010-1111-2222'

>>> price * tax
9.6
```

# 변수

- 첫 번째 문자에 숫자를 사용하지 않는다

```
>>> value = 100
>>> _value = 300
>>> 2value = 500
SyntaxError: invalid syntax
>>>
```

- 예약어를 사용하지 않는다

## 예약어

FALSE	None	TRUE	and	as	assert	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		

# 자료형 - 불린

- 불린(Boolean)이란?
  - ‘참’(True)과 ‘거짓’(False) 둘 중 한 가지 값을 취하는 논리 자료형
  - 논리 연산자(or, and not) 혹은 비교 연산자(>, <, == 등)을 활용

```
>>> x = True
>>> type(x)
<class 'bool'>
>>> x
True
```

```
>>> y = False
>>> type(y)
<class 'bool'>
>>> y
False
```

```
>>> x == y
False
```

# 자료형 - 불린

- 아래 명령을 Python Shell에서 실행하여 보세요

```
>>> True and False
```

```
>>> True or False
```

```
>>> not True
```

```
>>> not False
```

```
>>> 1 and 0
```

```
>>> 1 or 0
```

```
>>> not 1
```

```
>>> not 0
```

# 자료형 - 불린

- 아래 명령을 Python Shell에서 실행하여 보세요

```
>>> bool(0)
```

```
>>> bool(0.0)
```

```
>>> bool(1)
```

```
>>> bool(-1)
```

```
>>> bool(10)
```

```
>>> bool("")
```

```
>>> bool([1, 2])
```

```
>>> bool()
```



# Boolean Expression

>>> True and False

>>> 10<5 and 10>9

>>> True or False

>>> (10<5) and (10>9)

>>> not(True or False)

>>> 20 + 1 < 2

>>> 10=<20

>>> 'apple'>'Apple'

# 자료형 - 불린

- in / not in 연산자
  - 특정 항목이 속해 있는지 존재 여부를 확인

```
>>> 2 in [1, 2, 3]
```

```
>>> 1 in ['1', '2', '3']
```

```
>>> 'a' in ['a', 'b', 'c']
```

```
>>> a in ['a', 'b', 'c']
```

# 자료형 - 숫자

- 정수(integer)와 실수(float)
  - 정수와 실수는 숫자 자료형 중 가장 많이 사용됨
  - 사칙연산이 가능(산술연산자 활용)

>>> x = 8

>>> x + 2

>>> x - 2

>>> x / 2

>>> x % 2

>>> x \*\* 2

```
>>> x = 8
>>> x = x + 2
>>> x
10
>>> x += 2
>>> x
12
```

# Practice 1-1 🧐

- 정수와 실수 연산
  - 정수형 변수 a에 10을 할당
  - a를 2로 나눈 후 그 결과 값을 다시 a에 할당
  - a의 세제곱을 구한 후 그 결과 값을 변수 c에 할당
  - a와 c를 합한 후 그 결과 값을 출력
- 실수형 변수 b에 10.0을 할당
- b에 3을 곱한 후 10을 빼고 그 결과 값을 변수 d에 할당
- d를 4로 나눈 나머지를 출력

130.0  
0.0

# 자료형 - 문자열

- 문자열(string)
  - 문자(텍스트) 형태의 데이터를 담기 위한 자료형
  - 작은 따옴표 혹은 큰 따옴표 사용

```
>>> s1 = 'Apple'
>>> type(s1)
<class 'str'>
>>> s1
'Apple'
>>> s2 = ' '
>>> type(s2)
<class 'str'>
>>> s2
' '
>>> len(s1)
5
>>> len(s2)
1
```

# 자료형 - 문자열

- 문자열 결합(concatenation)

```
>>> s = s1 + s2
>>> s
'Apple '
```

- 인덱싱(indexing)

```
>>> s[0]
'A'
>>> s[1]
'p'
>>> s[2]
'p'
>>> s[-2]
'e'
>>> s[-1]
' '
```

# 자료형 - 문자열

- 분할(slicing)

```
>>> s3 = 'Hello Python'
>>> s3[0:5]
'Hello'
>>> s3[:5]
'Hello'
>>> s3[-6:]
'Python'
>>> s3[2:7]
'ello P'
```

- 서식설정(formatting)

- '{}'.format(args)

```
>>> s1 = '{} is fun!'.format('Python')
>>> s1
'Python is fun!'
>>> s2 = '{} , {}, and {} are fun!'.format('Python', 'Ruby', 'Java')
>>> s2
'Python, Ruby, and Java are fun!'
```

# String Basic Operations

```
>>> x = 'pop'
```

```
>>> y = 'corn'
```

```
>>> z = x + y
```

```
>>> z
```

```
>>> z[0]
```

```
>>> len(z)
```

```
>>> z[2:4]
```

```
>>> z.index('c')
```

```
>>> addr = 'abc@mobis.co.kr'
```

```
>>> atIndex = addr.index('@')
```

```
>>> length = len(addr)
```

```
>>> username = addr[0:atIndex]
```

```
>>> hostname = addr[atIndex+1:length]
```



# String Basic Operations

```
>>> x = 'pop'
```

```
>>> y = ' pop-corn '
```

```
>>> x.find('p')
```

```
>>> x.rfind('p')
```

```
>>> x.replace('p', 'i')
```

```
>>> y.strip(' ')
```

```
>>> y.split('-')
```

```
>>> x = 'banana'
```

```
>>> y = 'y2k'
```

```
>>> x.isalpha()
```

```
>>> x.isdigit()
```

```
>>> x.upper()
```

# Practice 1-2 🧐

- 문자열 결합과 분할

- 'I', 'love', 'you' 세 개의 문자열을 변수 str1, str2, str3에 순서대로 할당
- 세 개의 문자열을 결합해서 'I love you'라는 문자열을 만든 후 이를 변수 str4에 할당하고 str4를 출력
- 분할 연산자를 사용해서 str4로부터 'I love' 문자열만 추출한 후 그 결과 값을 다시 str4에 할당
- 결합 연산자를 사용해서 str4와 'programming'이라는 문자열을 결합한 후 그 결과 값인 'I love programming'을 새로운 변수인 str5에 할당한 후 str5를 출력

```
I love you  
I love programming
```

# Practice 1-3 🧐

- 식사 비용 계산

이번 휴가 때 가족들과 함께 해외 여행을 가기로 했다. 그 여행지 식당에서는 음식을 주문하면 음식 가격에 tip과 세금이 부과된다고 한다. 미리 주문할 음식의 가격을 확인해 보니 다음과 같았다.

- 음식 총 가격 : \$157.50
- 세율 : 음식 총 가격의 8.875%
- tip : 음식과 세금을 합친 가격의 15%

우리 가족이 저녁으로 지불해야 할 비용이 얼마일까?

The total cost is 197.20

# 자료형 - 리스트

## ● 리스트

- 순서를 가지는 0개 이상의 객체를 참조하는 배열형
- 리스트는 생성된 후 내용 변경이 가능(삽입, 변경, 삭제)
- 대괄호 [ ] 로 표현

```
>>> L1 = [1,2,3]
>>> type(L1)
<class 'list'>
>>> L1
[1, 2, 3]
>>> L2 = [[1, 2], [3, 4], [5, 6]]
>>> type(L2)
<class 'list'>
>>> L2
[[1, 2], [3, 4], [5, 6]]
>>> L2[0]
[1, 2]
>>> L3 = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
>>> L3[0]
[[1, 2], [3, 4]]
>>> L3[0][1]
[3, 4]
```

```
>>> len(L1)
3
>>> len(L2)
3
>>> len(L3)
2
```

# 자료형 - 리스트

인덱스로 리스트 항목에 접근, 수정

```
>>> L4 = [1, 2, 3]
>>> L4[0] = 0
>>> L4
[0, 2, 3]
```

슬라이싱으로 리스트 객체 수정

```
>>> L5 = [2, 4, 6, 7, 9]
>>> L5[:3] = [8, 10]
>>> L5
[8, 10, 7, 9]
```

del 명령문으로 리스트 항목 제거

```
>>> L6 = ['a', 'b', 'c', 'd', 'e']
>>> del L6[1]
>>> L6
['a', 'c', 'd', 'e']
```

# 자료형 - 리스트

- 리스트 메소드

- append()
- sort()
- count()

```
>>> L7 = [1, 2, 3]
>>> L7.append(4)
>>> L7
[1, 2, 3, 4]
```

```
>>> L8 = [4, 7, 2, 3, 1]
>>> L8.sort()
>>> L8
[1, 2, 3, 4, 7]
```

```
>>> L9 = ['b', 'e', 'd', 'a', 'b', 'f']
>>> L9.count('b')
2
>>> L9.sort()
>>> L9
['a', 'b', 'b', 'd', 'e', 'f']
```

# List Operations

```
>>> lst = [1, 2, 3, 4, 5]
```

```
>>> lst
```

```
>>> lst[0]
```

```
>>> lst[1] = 2.5
```

```
>>> lst
```

```
>>> lst.append(6)
```

```
>>> lst
```

```
>>> lst.insert(3, 3.5)
```

```
>>> lst
```

```
>>> del lst[3]
```

```
>>> lst
```

```
>>> lst2 = [2, 4, 6]
```

```
>>> new_list = lst + lst2
```

```
>>> new_list
```

# List Operations

```
>>> lst = [5, 3, 8, 2, 4]
```

```
>>> lst
```

```
>>> lst[0]
```

```
>>> lst[-1] = 6
```

```
>>> lst[0:3]
```

```
>>> lst[:3]
```

```
>>> lst[3:]
```

```
>>> lst.append(9)
```

```
>>> lst
```

```
>>> lst.insert(1, 7)
```

```
>>> lst
```

```
>>> lst.index(8)
```

```
>>> lst.pop()
```

```
>>> lst.remove(5)
```

```
>>> lst
```

```
>>> lst.reverse()
```

```
>>> lst
```

```
>>> lst.sort()
```

```
>>> lst
```



# Practice 1-4 🧐

- 리스트 생성 및 수정
  - 다음 항목(객체)을 포함하는 리스트 생성  
1, 7, [4, 9], 'f', ('a', 'b')
  - 'g' 항목을 리스트의 마지막에 추가
  - 분할 연산자를 사용해서 리스트의 마지막 두 항목만 가지고 있는 리스트를 생성

```
[1, 7, [4, 9], 'f', ('a', 'b'), 'g']  
[('a', 'b'), 'g']
```

# Practice 1-5 🧐

- 2차원 리스트

- 알파벳 대문자(A~O)로 이루어진 3행 5열의 2차원 리스트를 생성하고 각 행을 소문자로 바꾸어 출력
  - (3, 5) 크기의 2차원 리스트를 정의
  - 행 별로 한 줄에 소문자로 출력
  - .lower() 활용

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o

# 자료형 - 튜플

## ● 튜플

- 리스트와 유사하지만 튜플은 생성 후 내용 변경이 불가
- 소괄호 ( ) 로 표현

```
>>> t1 = (1, 2, 3)
>>> type(t1)
<class 'tuple'>
>>> t1
(1, 2, 3)
```

- 소괄호 없이도 생성할 수 있음

```
>>> t2 = 'a', 'b', 1, 2
>>> type(t2)
<class 'tuple'>
>>> t2
('a', 'b', 1, 2)
```

```
>>> t3 = (t1, t2)
>>> t3
((1, 2, 3), ('a', 'b', 1, 2))
```

# 자료형 - 튜플

- 튜플의 길이는 튜플이 가지고 있는 항목의 수와 같은 의미
- 튜플은 리스트로 형변환이 가능하고 리스트도 튜플로 형변환이 가능

```
>>> atuple = (1, 2, 3)
>>> alist = list(atuple)
>>> alist
[1, 2, 3]
>>> atuple = tuple(alist)
>>> atuple
(1, 2, 3)
```

# 자료형 - 튜플

- 인덱싱과 슬라이싱
  - 문자열과 비슷
  - 배열형인 문자열, 리스트, 튜플에 적용

```
>>> t4 = ('a', 'b', 'c', 1, 2, 3)
>>> t4[0]
'a'
>>> t4[3:]
(1, 2, 3)
```

```
>>> t5 = ('a', 'b', 'c', (1, 2, 3))
>>> t5[0]
'a'
>>> t5[-1]
(1, 2, 3)
>>> t5[:2]
('a', 'b')
```

# Practice 1-6 🧐

- 튜플 분할 생성 및 멤버십 연산자
  - 아래 객체를 포함하는 튜플을 생성해서 변수 atuple에 할당  
1, 5, (2, 3), 'green', ['드럼', '기타']
  - 분할 연산자를 사용해서 atuple의 마지막 두 항목만 가지고 있는 튜플 btuple을 생성
  - (2, 3)이 atuple에 포함되어 있는지 확인
  - 5가 btuple에 들어 있는지 확인

True  
False

# 자료형 - 셋

## ● 셋

- 셋은 리스트와 유사하지만 항목의 순서가 없고, 중복된 값을 포함할 수 없음
- 중괄호 { } 로 표현
- 셋은 리스트나 튜플로 형변환이 가능하며 그 반대도 가능

```
>>> s1 = {1, 2, 3, 4, 3, 2}
>>> type(s1)
<class 'set'>
>>> s1
{1, 2, 3, 4}
```

```
>>> s2 = {4, 5, 2}
>>> t = tuple(s2)
>>> t
(2, 4, 5)
>>> L = list(s2)
>>> L
[2, 4, 5]
```

```
>>> s2 = set(L)
>>> s2
{2, 4, 5}
```

# 자료형 - 셋

- 셋 메소드

- add()
- remove()
- update()

```
>>> s3 = {1, 10, 2}  
>>> s3.add(5)  
>>> s3.add(1)  
>>> s3  
{1, 10, 2, 5}
```

```
>>> s4 = {2, 4, 5}  
>>> s4.remove(5)  
>>> s4  
{2, 4}
```

```
>>> s5 = {2, 7, 3}  
>>> s5.update([2, 4, 5])  
>>> s5  
{2, 3, 4, 5, 7}
```

update() 메소드는 하나 이상의 항목을 셋에 추가함



# Practice 1-7 🧐

- 셋 생성 및 연산
  - 아래 객체를 포함하는 셋을 생성해서 변수 set1에 할당  
2, 4, 5, 7
  - 아래 객체를 포함하는 셋을 생성해서 변수 set2에 할당  
1, 2, 7, 9
  - set1과 set2의 셋 결합(합집합) 연산 결과 출력
  - set1과 set2의 셋 교차(교집합) 연산 결과 출력

```
{1, 2, 4, 5, 7, 9}  
{2, 7}
```

# 자료형 - 딕셔너리

## ● 딕셔너리

- 딕셔너리의 각 항목은 키-값(key-value) 쌍으로 구성되어 있음
- 딕셔너리의 객체는 순서가 없고 key를 통해 value에 접근할 수 있음

```
>>> d1 = {1: 'red', 2: 'blue', 3: 'yellow', None: ''}
>>> d1[1]
'red'
>>> d1[3]
'yellow'
>>> d1[None]
''
```

```
>>> d2 = {'a': 1, 'b': 2, 'c': 3}
>>> d2
{'a': 1, 'b': 2, 'c': 3}
>>> d2['d'] = 4 항목 추가
>>> d2
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> del d2['a'] 항목 삭제
>>> d2
{'b': 2, 'c': 3, 'd': 4}
```

# 자료형 - 딕셔너리

- 딕셔너리 메소드

- items()
- keys()
- values()

```
>>> d3 = {'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona'}  
>>> d3_items = d3.items()  
>>> d3_items  
dict_items([('AL', 'Alabama'), ('AK', 'Alaska'), ('AZ', 'Arizona')])
```

```
>>> d3_keys = d3.keys()  
>>> d3  
{'AL': 'Alabama', 'AK': 'Alaska', 'AZ': 'Arizona'}
```

```
>>> d3_values = d3.values()  
>>> d3_values  
dict_values(['Alabama', 'Alaska', 'Arizona'])
```

# Practice 1-8 🧐

- 딕셔너리 생성 및 수정

- 아래 키-값(key-value) 쌍으로 되어있는 딕셔너리를 생성해서 변수 d에 할당한 후 d를 출력(값은 튜플로)

Key	Value
"even"	(2, 4, 6, 8, 10)
"odd"	(1, 3, 5, 7, 9)
"prime"	(2, 3, 5, 7)

- 아래 키-값 쌍을 d의 항목으로 추가한 후 d를 출력

키: "all"

값: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

- 키 값이 'odd'인 항목을 삭제한 후 d를 출력

```
{'even': (2, 4, 6, 8, 10), 'odd': (1, 3, 5, 7, 9),  
'prime': (2, 3, 5, 7)}  
{'even': (2, 4, 6, 8, 10), 'odd': (1, 3, 5, 7, 9),  
'prime': (2, 3, 5, 7), 'all': (1, 2, 3, 4, 5, 6, 7,  
8, 9, 10)}  
{'even': (2, 4, 6, 8, 10), 'prime': (2, 3, 5, 7), '  
all': (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)}
```

# Practice 1-9 🧐

- 딕셔너리 항목 필터링

- 아래 직원들 중 연봉이 50,000 이상인 직원들의 이름을 딕셔너리를 활용해 출력
- 직원 연봉 정보
  - David : 30000
  - John : 50000
  - Andrew : 45000
  - Rita : 70000
  - Michale : 10000

```
John's salary is 50000  
Rita's salary is 70000
```

# Type Conversion

```
>>> int(10.9)
```

```
>>> int('10')
```

```
>>> int('10.9')
```

```
>>> float('10.9')
```

```
>>> float(10.9)
```

```
>>> ord('A')
```

```
>>> ord('Z') - ord('A')
```

```
>>> ord('A') > ord('a')
```

```
>>> chr(65)
```

```
>>> chr(ord('A'))
```

```
>>> chr(108) + chr(111) + chr(118) + chr(101)
```

# Type Conversion

- 다음 명령어들의 차이를 확인하여 보세요

```
>>> format(15, 'X')
```

```
>>> format(15, '2X')
```

```
>>> format(15, '02X')
```

```
>>> format(65, 'b')
```

```
>>> format(65, '08b')
```

# Type Conversion

- 체질량 지수(BMI)를 계산하는 다음 코드를 작성하여 보세요

```
>>> weight = float(input('Enter weight(kg) : '))
```

```
>>> height = float(input('Enter height(m) : '))
```

```
>>> bmi = weight / (height ** 2)
```

```
>>> print('Your BMI: ', format(bmi, '.1f'))
```