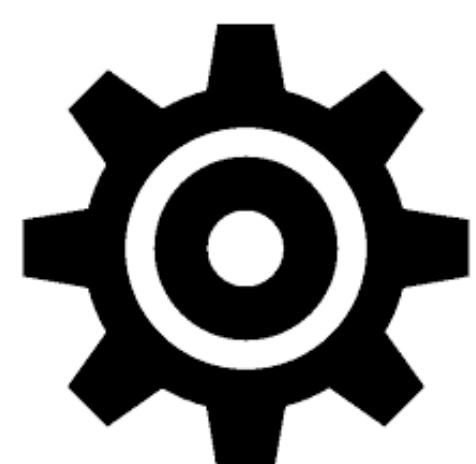




CHAPTER 5

기본자료형

Basic Data Types



박진수 교수
서울대학교·경영대학
jinsoo@snu.ac.kr



학습 목차

프로그램 구성 단위

파이썬 자료형

불린형

숫자형

- ▶ 정수형
- ▶ 실수형

문자열형

프로그램 구성 단위

Program Units





식별자

- 변수, 함수, 클래스 이름 등

예약어

- 파이썬 키워드(명령어)

리터럴(literal)

- 읽혀지는 대로 쓰여있는 값 그 자체

- e.g., 3, -5.7, 'apple'

- 식별자는 프로그램이 실행되는 동안 다양한 값을 가질 수 있는 이름이지만, 리터럴은 이름이 아니라 값 그 자체

- e.g., x = 3 (여기서 x는 식별자, 즉 변수이고, 3은 리터럴)

표현식(expression)

- 새로운 데이터 값을 생성하거나 계산하는 코드 조각

- e.g., 2 + 5, 1 + 2 + 3 * (8 ** 9) - sqrt(4.0), 'apple' + 'box', round(81.5), max(3, -2, 12, 15, -37, 52, 78, 94)



프로그램 구성 단위

● 명령문(statement)

- 특정한 작업을 수행하는 코드 전체
- e.g., `print(3 + 4)`, `if`-문 등



표현식은 값을 생성하는 일부분이고 명령문은 특정 작업을 수행하는 코드 전체

● 함수(function)

- 특정 명령을 수행하는 코드 묶음(주로 여러 명령문이 모여 함수가 됨)

● 모듈(module)

- 함수/클래스의 모음 또는 하나의 프로그램을 구성하는 단위

● 패키지(package)

- 프로그램과 모듈 묶음
 - 프로그램 : 실행하기 위한 것
 - 모듈 : 다른 프로그램에서 불러와 사용하기 위한 것

● 라이브러리(library)

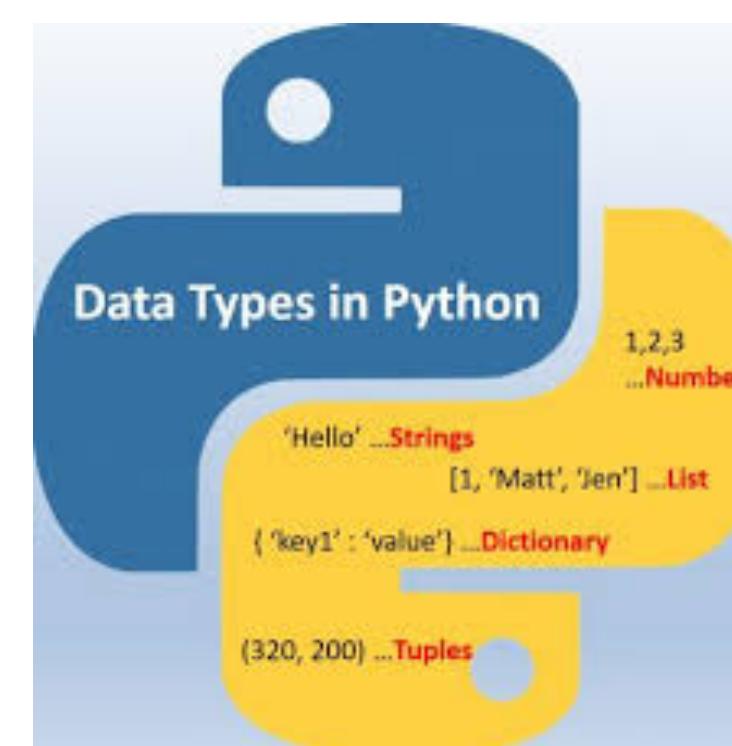
- 패키지 모음

```
"""This file is provided for educational purpose and distributed for class use.  
Copyright (c) by Jinsoo Park, Seoul National University. All rights reserved.  
  
File name.....: hello.py  
Description...: A very simple program which prints 'Hello World' using  
two variable concatenation.  
"""\n\n# The following code displays "Hello Python" in Korean on the screen  
print('안녕 파이썬') # print()함수를 호출  
  
# !!!!! END of hello.py !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

- 줄 앞이나 중간에 '#'를 붙이면 '#' 이후는 주석 처리된다
- 중간 중간 빈 줄을 넣으면 코드의 가독성을 높일 수 있다
- 파이썬 코드 (명령문)
 - ▶ `print()` 함수 : '안녕 파이썬'이란 문자열 전달인자를 출력하는 함수
- 도움말(help)
 - ▶ 대화형 모드에서 `help()`를 입력하면 파이썬 도움말 유틸리티를 실행할 수 있다
 - ▶ `help()` 전달인자에 특정 모듈이나 키워드, 주제의 이름을 입력하면 해당하는 도움말을 얻을 수 있다
 - >>> `help(print)`

파이썬 자료형

Python Data Types

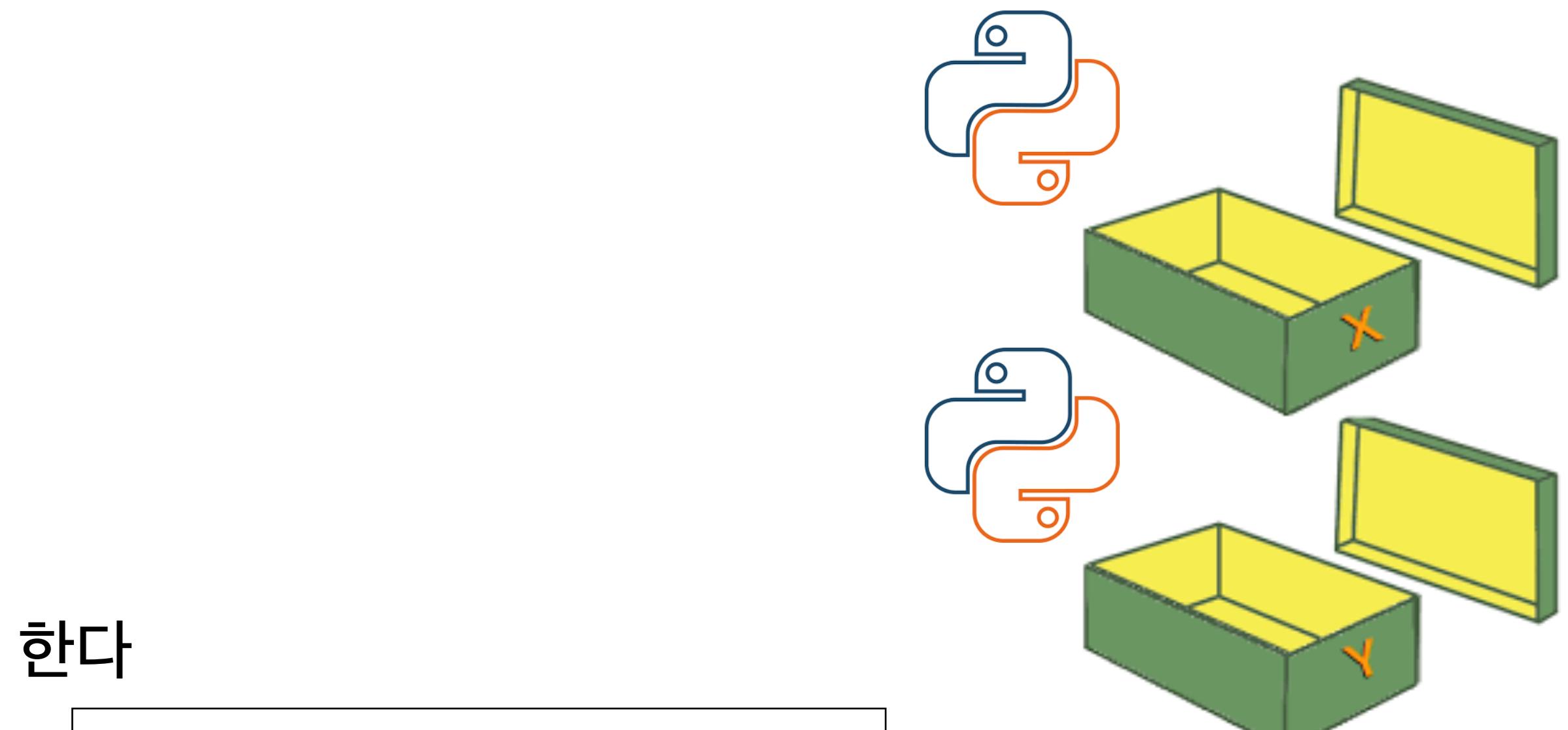




- 자료형(data type)은 다음과 같은 정보를 담고 있다
 - 해당 자료형이 가질 수 있는 가능한 값의 종류
 - 해당 자료형이 참조하는 데이터(객체)가 컴퓨터 메모리에 저장되는 방식
 - 해당 자료형으로 실행할 수 있는 수학적, 관계적, 논리적 명령들

- 어떤 명령어들은 자료형에 따라 다른 결과 값을 돌려주기도 한다
 - + 연산자
 - 정수끼리 더하면 정수의 합을 반환하지만
 - 문자열끼리 더하면 문자열 결합을 반환한다

 - * 연산자
 - 정수끼리 곱하면 정수의 곱한 결과 값을 반환하지만
 - 문자열을 곱하면 곱한만큼 반복한 문자열 결과 값을 반환한다



```
1 + 2 # 3  
'1' + '2' # '12'
```

```
2 * 5 # 10  
'2' * 5 # '22222'
```



- 자료형은 다음 중 하나의 속성을 가짐

- **불변성**(immutable) : 생성한 후 내용의 변경이 불가

- ◆ e.g., 불린형, 정수형, 실수형, 문자열형, 튜플형 등

- **가변성**(mutable) : 생성한 후 내용의 변경이 가능

- ◆ 리스트형, 세트형, 딕셔너리형 등

- 순회형(iterable)

- 한 개 이상의 값(객체)을 동시에 가지고 있는 자료형

- ◆ 즉, 여러 개의 요소(element)로 이루어져 있다

- ◆ 예) 리스트(list): [1, 2, 3, 4, 5]

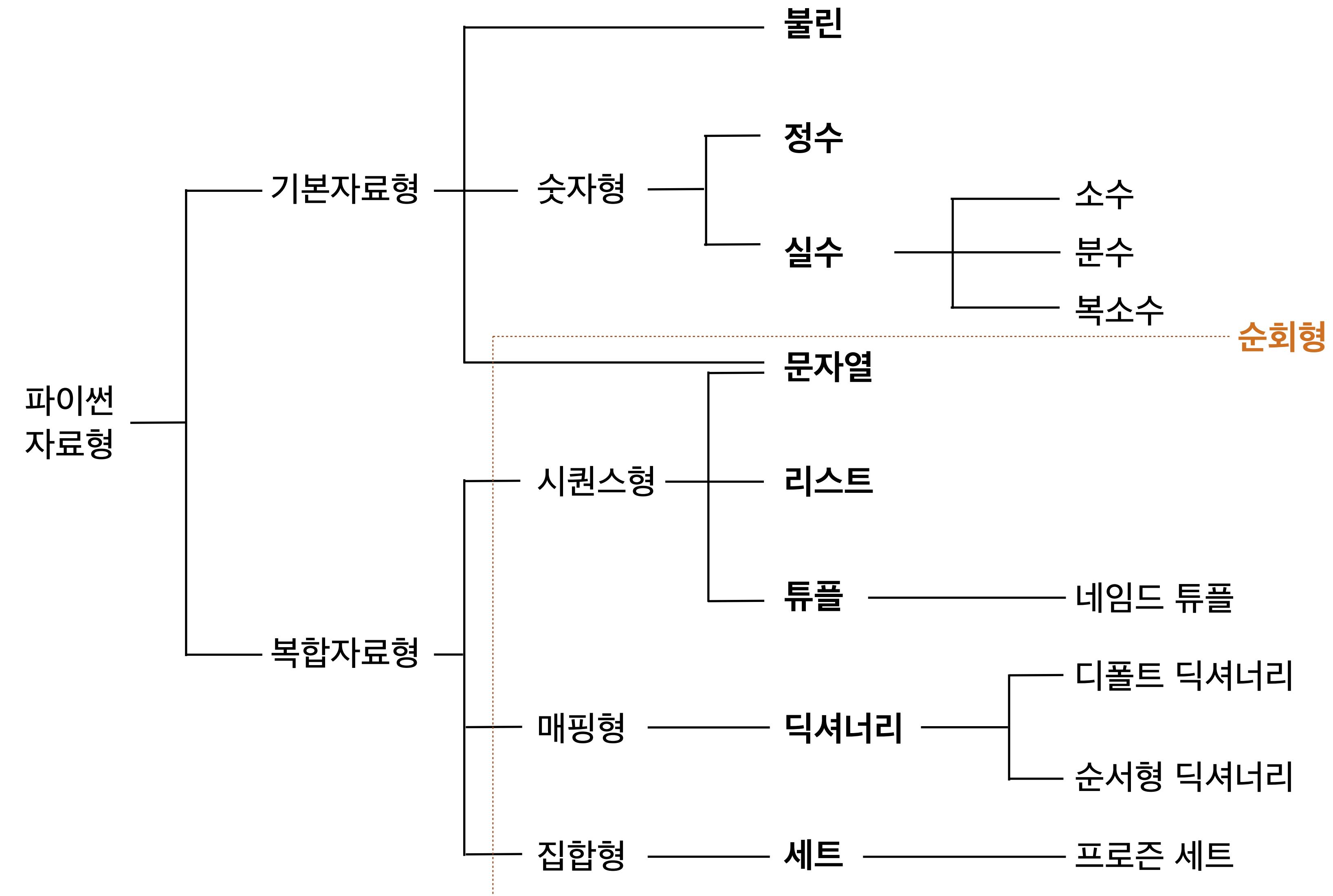
- 여러 개의 요소를 하나씩 접근하는 순회(iterate)가 가능

- 문자열은 기본 자료형이지만 여러 개의 요소로 나누는 것이

- ◆ 예) 'Python' → 'P', 'y', 't', 'h', 'o', 'n'



파이썬 자료형 분류





적절한 자료형을 선택하는 방법

● 첫번째 고려 사항

- 이 숫자로 수학적 계산을 할 것인가?

- Yes

- No

- 예시

- 우편번호?

- 제품 가격?

- 고객수?

- 주민번호?

- 전화번호?

● 두번째 고려 사항

- 여러 개의 자료를 저장할 것인가?

- Yes

- No

● type(객체) 클래스

- 객체(object)의 자료형(타입)을 알 수 있음
- 대화형 모드
- 함수에서 전달인자를 받아 처리할 경우
- 테스트나 디버깅 단계

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

type(5)
<class 'int'>

type(12.345)
<class 'float'>

type('12.345')
<class 'str'>

type('안녕 파이썬')
<class 'str'>

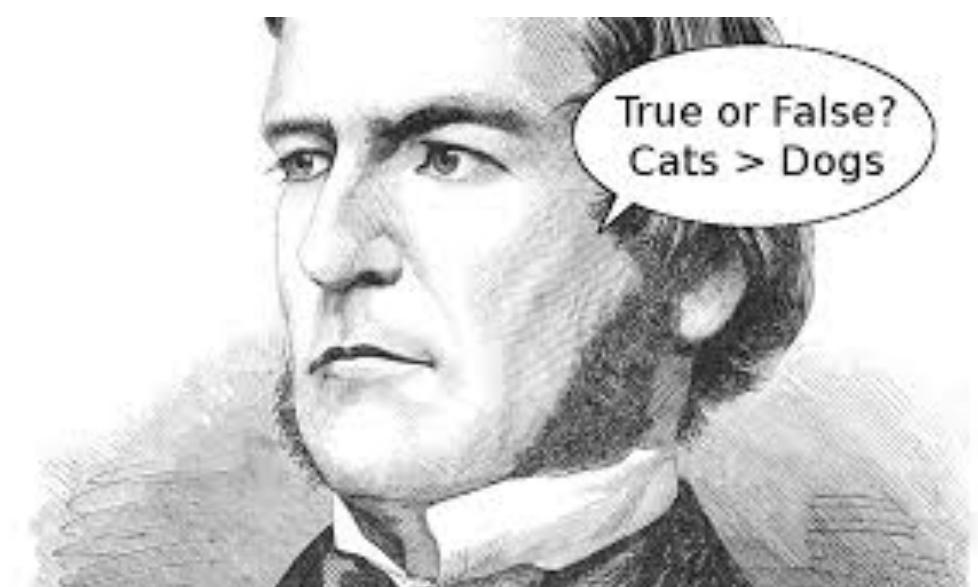
type(print)
<class 'builtin_function_or_method'>

type(help)
<class '_sitebuiltins._Helper'>

type(list)
<class 'type'>
```

불린형

Boolean Type





불린(Boolean)이란?

- ⦿ 불변자료형(immutable)
- ⦿ '거짓'(False)과 '참'(True) 두 가지 값 중 한 개만 취할 수 있는 논리 자료형

⦿ 가능한 값

⦿ 거짓(False)

- ⦿ False, 0, 0.0, None, 빈 문자열(' '), 빈 복합자료형([], (), {}, set())

⦿ 참(True)

- ⦿ 그 외 나머지
 - True, 0이 아닌 정수 등

⦿ 참, 거짓이 숫자로 표현될 경우

⦿ False → 0

⦿ True → 1

⦿ 주로 다음에서 사용

⦿ 논리 연산 : not, and, or

⦿ 비교 연산(관계 연산) : 같다(==), 다르다(!=), 크다(>), 작다(<), 크거나 같다(>=), 작거나 크다(<=)

```

11 - False
11 + True
int(False)
int(True)

```



bool() 클래스 특정 데이터가 True인지 False인지를 검증

bool(0)

bool(0.0)

bool(1)

bool(-1)

bool(10)

bool(0.1)

bool('')

bool([])

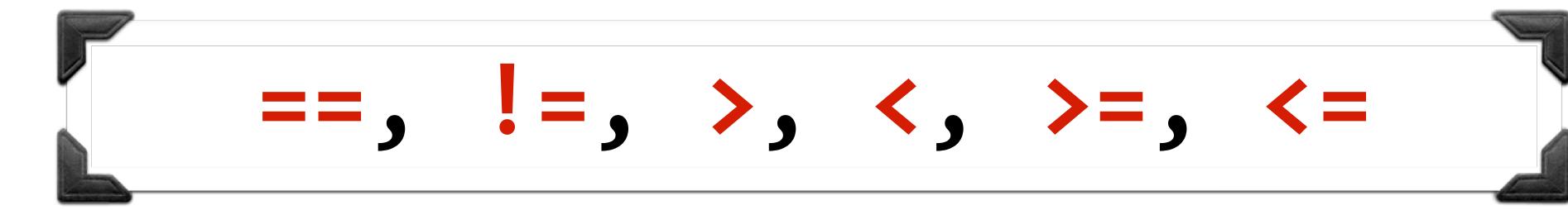
bool([1, 2])

bool(())



비교 연산자

비교 연산자 또는 관계 연산자



==, !=, >, <, >=, <=

- 양쪽에 있는 값을 서로 비교
- 결과 값을 참(True) 또는 거짓(False) 형태로 반환
- 숫자 또는 문자열을 비교
 - 문자열은 유니코드 값을 기준으로 비교
 - 객체의 메모리 주소(객체참조)가 아닌 객체의 값을 비교
- 프로그래밍에서 비교 연산자는 수학에서 사용될 때와 조금 다른 형태로 표현
 - e.g., ==, !=

연산자	설명
$x == y$	▶ x 와 y 가 같은가?
$x != y$	▶ x 와 y 가 다른가?
$x < y$	▶ x 가 y 보다 작은가?
$x <= y$	▶ x 가 y 보다 작거나 같은가?
$x > y$	▶ x 가 y 보다 큰가?
$x >= y$	▶ x 가 y 보다 크거나 같은가?



예시 : 비교 연산

'a' == 'A'

'a' < 'A'

'a' <= 'z'

5 != 5.0

5 > 5.0

5 >= 5.0



논리 연산자(logic operations) : 불린 연산에 사용

문법	설명
not x	▶ x 가 False인 경우 True를 반환하고 x 가 True이면 False를 반환 - not은 x 의 반대를 반환
x and y	▶ 최소평가(short-circuit) 연산자로 x 와 y 가 모두 참인 경우에만 True를 반환 - y 는 x 가 True인 경우에만 계산
x or y	▶ 최소평가(short-circuit) 연산자로 x 또는 y 둘 중 하나라도 참인 경우 True를 반환 - y 는 x 가 False인 경우에만 계산

```
if True or False:  
    print('Hello')
```

```
if False and True:  
    print('Hello')
```



not True

not False

False and True

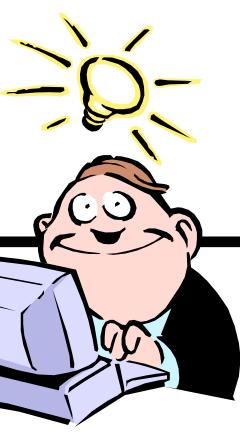
True and True

True or False

False or False



최소평가(short-circuit) 연산을 눈으로 확인할 수는 없을까?



0 and 5

1 and 5

5 or 1

0 or -1



한 번에 여러 개의 논리 연산자를 사용할 수 없을까?



가능하다. 하지만 논리 연산자 사이에 우선 순위가 있다.

아래 코드를 실행한 결과는?

False and not True or True



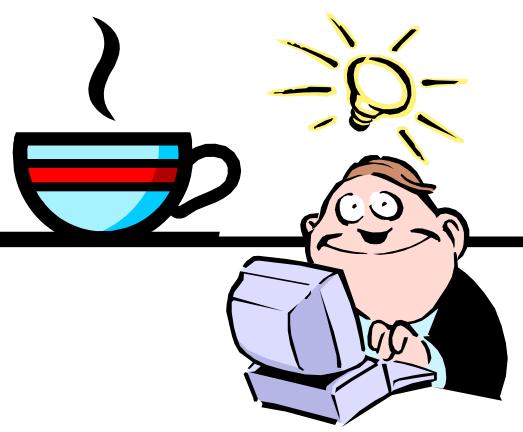
논리 연산자는 왼쪽에서 오른쪽으로 평가되지 않는다

즉, 연산 우선 순서가 있다

not > and > or

True or not False and False

not not True or False and not True



in / not in 연산자

- 특정 항목이 속해 있는지 존재 여부를 확인
- ✓ 복합자료형 리스트에서 다룰 예정

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

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

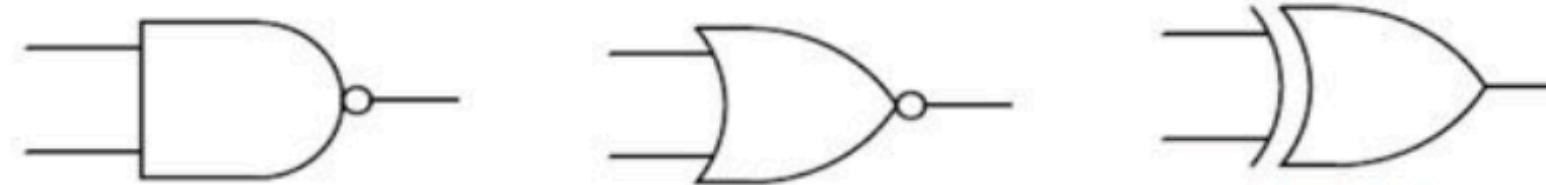
```
'a' in ['a', 'b', 'c']
```

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



● NAND, NOR, XOR 게이트 구현

- 논리 회로에서의 대표적인 논리 게이트인 NAND, NOR, XOR 게이트를 구현한다
- 사용자로부터 0 혹은 1인 x, y 값을 입력받아, 입력에 대한 각 게이트의 출력 값을 확인할 수 있어야 한다



NAND

NOR

XOR

A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

● 실행 결과 예시

x 값을 입력하세요(0 혹은 1): 0
y 값을 입력하세요(0 혹은 1): 0

x NAND y -> 1
x NOR y -> 1
x XOR y -> 0

x 값을 입력하세요(0 혹은 1): 1
y 값을 입력하세요(0 혹은 1): 1

x NAND y -> 0
x NOR y -> 0
x XOR y -> 0

x 값을 입력하세요(0 혹은 1): 0
y 값을 입력하세요(0 혹은 1): 1

x NAND y -> 1
x NOR y -> 0
x XOR y -> 1

숫자형

Numeric Data Types



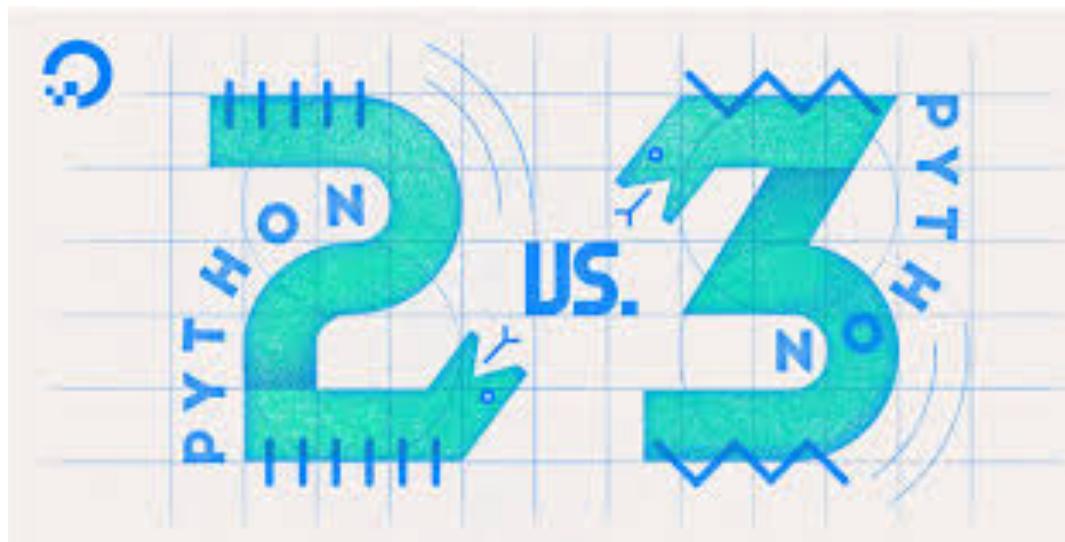
● 숫자형이란?

- 일반적으로 수치를 나타내고 수치 연산을 수행하기 위해 사용하는 자료형
- 정수(integer)와 실수(float)는 숫자 자료형 중 가장 많이 사용하는 자료형



정수형

Integer Type



- 정수(integers)란?

- 불변자료형(immutable)

- 소수점이 없는 숫자(digits) : `int`

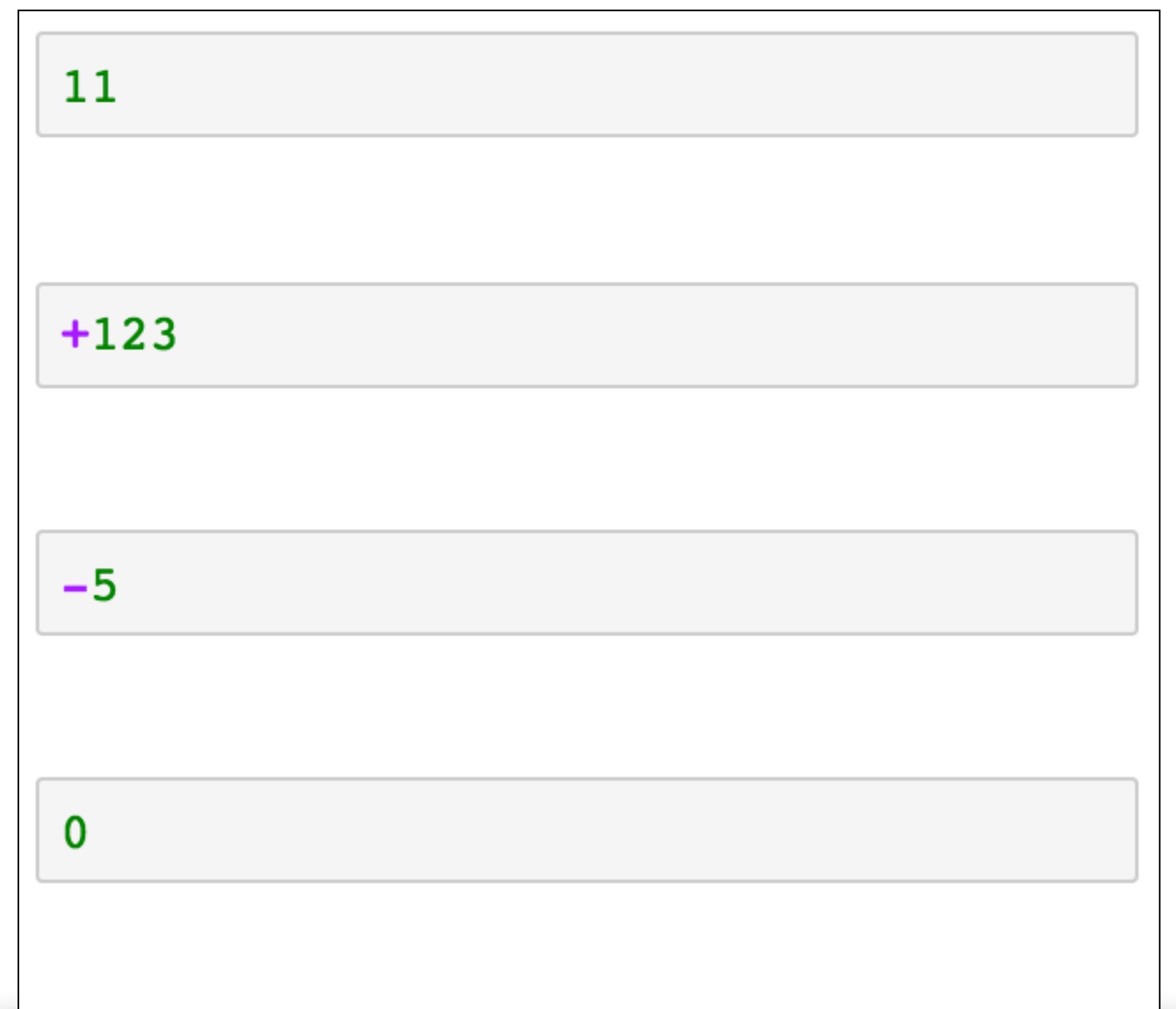
- 숫자 : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- 표현법

- 기본적으로 10진수로 표현됨

- 정수의 크기

- 타 언어와는 달리, 정수의 크기가 컴퓨터의 성능(메모리)에 의해 한정됨





더하기 / 빼기 / 곱하기

9 + 5 # 더하기 연산

9 - 5 # 빼기 연산

9 * 5 # 곱하기 연산



9 / 5 # 나누기 연산

9 // 5 # 몫 구하기 연산

9 % 5 # 나머지 구하기 연산



```
1 2 ** 3 # 거듭제곱 연산( $2 ** 3 == 2 \times 2 \times 2$ )
```

```
1 2 ** -1 # 음수 제곱은?
```

```
1 123456789 ** 0 # 모든 수의 0제곱은?
```



1 + 2.0

3.0 - 3

1 * 1.0



한 번에 여러 개의 연산을 할 수 없을까?

예) x 더하기 y 곱하기 z



Lab : 정수 연산



● 다음 수식을 파이썬으로 계산하기

- ➊ $5 \times 3 - 11 \div 2$
- ➋ $11 \times 9 \div 3 + 25$
- ➌ $910520 - 4403505 \div 880701$
- ➍ $5^3 - 8 \% 9$
- ➎ 위 식에서 %는 나머지를 구하는 연산

● 실행 결과 예시

9.5
58.0
910515.0
117



아래 코드를 실행한 결과는?

```
x = 2  
y = 3  
z = 5  
  
print(x + y * z)
```



피연산자	부호
괄호	()
지수	**
곱셈 / 나눗셈	* , /
덧셈, 뺄셈	+ , -



예시 : 연산 순서

```
x = 2 + 3 * 5  
x # x의 값은 무엇인가?
```

```
y = (2 + 3) * 5  
y # y의 값은 무엇인가?
```



Lab : 우선 순위와 정수 연산



● 다음 수식을 파이썬으로 계산하기

- ➊ $5 \times (3 - 11 \div 2)$
- ➋ $9 \times (8 + (10 - 6) \div 2)$
- ➌ $700 - (365 - 5 \times 3) \times 2$
- ➍ $((5^3 - 5 \times 5) \% 5 + 3)^3$

➎ 위 식에서 %는 나머지를 구하는 연산

● 실행 결과 예시

-12.5
90.0
0
27



좌변의 변수값 대체

일반식

$x = x \text{ operator } y$

증강 할당 연산식

$x \text{ operator=} y$

```
x = y = 9
```

숫자형 변수 x의 값을 바꾼다.

```
x = x + 2
```

x # 이제 x는 9가 아니다.

더 간단한 방법은 증강 할당 연산자를 사용하는 것이다.

```
y += 2
```

y # 이제 y는 9가 아니다.



예시 : 증강 할당 연산과 변수값 대체

```
x = 5  
y = 9
```

```
x += y  
x
```

```
y -= x  
y
```

```
x *= 2  
x
```

```
x /= 2  
x
```

```
x //= 5  
x
```

```
y %= 2  
y
```

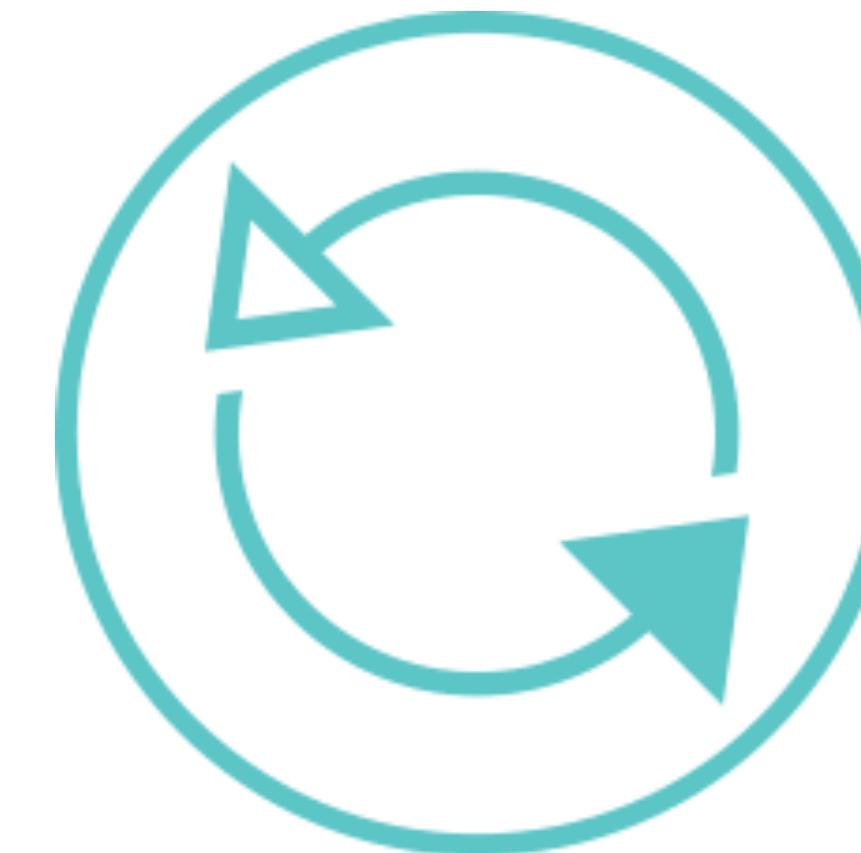


문법	설명
$x + y$	▶ x 와 y 를 더함; $[x = x + y]$ 는 $[x += y]$ 와 같음; x, y 중 하나가 실수면 실수를 반환
$x - y$	▶ x 에서 y 를 뺌; $[x = x - y]$ 는 $[x -= y]$ 와 같음; x, y 중 하나가 실수면 실수를 반환
$x * y$	▶ x 에 y 를 곱함; $[x = x * y]$ 는 $[x *= y]$ 와 같음; x, y 중 하나가 실수면 실수를 반환
x / y	<ul style="list-style-type: none"> ▶ x를 y로 나눔; $[x = x / y]$는 $[x /= y]$와 같음 ▶ <u>항상 실수</u>(float, 부동소수점) 값을 반환 <ul style="list-style-type: none"> - x나 y가 복소수일 경우 복소수형(complex) 값을 반환
$x // y$	<ul style="list-style-type: none"> ▶ x를 y로 나누고 소수점 이하는 버림(floor division); $[x = x // y]$는 $[x //= y]$와 같음 ▶ 소수점 이하를 버리기 때문에, 둘 다 정수인 경우에 <u>정수</u>(int) 값을 반환
$x \% y$	▶ x 를 y 로 나눴을 때 나머지 값을 반환; $[x = x \% y]$ 는 $[x %= y]$ 와 같음
$x ** y$	▶ x 의 y 제곱 값을 반환; $[x = x ** y]$ 는 $[x **= y]$ 와 같음; x, y 중 하나가 실수면 실수를 반환
$-x$	<ul style="list-style-type: none"> ▶ 부호 반전 <ul style="list-style-type: none"> - x가 0이 아닌 경우 부호를 반전 시킴 - x가 0이면 아무런 동작도 하지 않음
$+x$	▶ 아무런 동작도 하지 않음; 가끔 코드의 의미를 명확히 할 필요가 있을 때 사용



• int(객체)

- 객체를 정수로 변환
- 변환을 실패하면 ValueError 예외가 발생
- 변환한 숫자의 소수점 이하는 버림(반올림 하지 않음)





예시 : 정수형 변환

```
int()
```

```
int(10)
```

```
int(' -10 ')
```

```
int(12.999)
```

```
int('12.999')
```



실수형



Floating-Point Type

3.14159₂₆₅₃₅₈₉₇₉₃₂₃₈₄₆₂₆₄₃₃₈₃₂₇₉₅₀₂₈₈₄₁₉₇₁₆₉₃₉₉₃₇₅₁₀₅₈₂₈₆₀₉₂₉₀₁₉₂₃₄₈₈₃₄₉₉₁₃₄₄₉₄₃₈₂₀₄₁



- 실수(부동소수점수, floating-point numbers)란?

- 불변자료형(immutable)
- 소수점이 있는 숫자(digits) : **float**
 - 숫자 : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- 표현법
 - 소수점 또는 지수 표현법 사용
 - e.g., 0.0, .5, 4.7, -3.5, 7.9e-4





실수 연산

```
3 + 7.0      # 더하기 연산
```

```
.0 - .2      # 빼기 연산
```

```
5 * 1.2      # 곱하기 연산
```

```
3 * 7.0e+1   # 곱하기 연산
```

```
3.6 / 2      # 나누기 연산
```

```
3.6 // 2     # 몫 구하기 연산
```

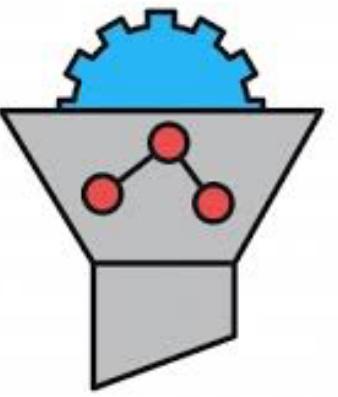
```
3.6 % 2      # 나머지 구하기 연산
```

```
2.0 ** 3      # 거듭제곱 연산(2.0 ** 3 == 2.0 x 2.0 x 2.0)
```



float(객체)

- 객체를 실수로 변환
- 변환을 실패하면 `ValueError` 예외가 발생



```
float()
```

```
float(123)
```

```
float('+1.2345')
```

```
float(' -12.345')
```

```
float(' -.4528')
```

```
float(-1e-3)
```

```
float('1e3')
```

```
float('3.123f')
```

```
float('-Infinity')
```

```
float('3.14') / float(-.1)
```

```
float('False')
```

```
float(False)
```

```
float('True')
```

```
float(True)
```



숫자 관련 함수와 클래스

Functions and Classes for Numbers





정수와 실수에서 자주 사용하는 함수

문법	설명
<code>abs(x)</code>	▶ x 의 절댓값을 반환
<code>divmod(x, y)</code>	▶ x 를 y 로 나눴을 때의 ‘몫’과 ‘나머지’, 총 2개의 정수를 반환
<code>pow(x, y)</code>	▶ $x^{**}y$ 와 같다
<code>pow(x, y, z)</code>	▶ $(x^{**}y) \% z$ 와 같다
<code>round(x, n)</code>	▶ n 이 양의 정수면 x 를 소수점 n 자리로 반올림하여 반환 ▶ n 이 음의 정수면 x 를 n 자리에서 반올림하여 반환 ▶ 반환되는 값의 자료형은 x 와 같다



예시 : 정수와 실수에서 자주 사용하는 함수

```
abs(-12)
```

```
divmod(13, 7)
```

```
pow(2, 3)
```

```
pow(2, 3, 5)
```

```
round(1.23546, 1)
```

```
round(1.23546, 2)
```

```
round(1.23546, 3)
```

```
round(123546, -2)
```

```
round(123546, -3)
```

```
round(123546.789, -3)
```



식사 비용 계산하기

이번 휴가 때 가족들과 함께 해외 여행을 가기로 했다. 이것 저것 준비하다 알게된 사실인데 외국 식당에서는 음식을 주문하면 음식 가격에 봉사료(tip)와 세금(meal tax)이 부과된다고 한다. 그래서 내가 출국하기 전에 미리 우리 가족 저녁을 위해 예약한 식당의 웹사이트를 방문해서 주문할 음식들의 가격을 미화로 알아보고 그 나라의 봉사료과 음식세금도 확인해 보니 다음과 같았다.

- 음식 총 가격 : \$137.50
- 세율 : 음식 총 가격의 8.875%
- 봉사료 : 음식과 세금을 합친 가격의 15%

여기서 주의할 점은 봉사료를 계산할 때 음식 가격의 15%가 아니라 음식과 세금을 합친 가격의 15%라고 한다. 그럼 우리 가족의 저녁으로 지불해야 할 비용이 얼마일까?

실행 결과 예시

172.16



잠깐...

휴대폰의 계산기나 파이썬을 이용해서 다음 내용을 따라 해보자.

1. 먼저 세 자리 숫자를 마음 속으로 생각한다.
2. 생각한 숫자에 7을 곱한다.
3. 그 숫자에 다시 11을 곱한다.
4. 그 숫자에 다시 13을 곱한다.
5. 어떤 결과가 나왔나??

$7 \times 11 \times 13 = 1,001$ 이다.

이제 왜 그런 결과가 나왔는지 알겠나요?

처음부터 1001을 곱했다면 결말을 예상하기 쉬웠겠죠!!!



프로그래밍을 할 때도 종종 이런 결정을 내려야 할 때가 있다.

한 줄에 여러 연산을 작성할 것인지

한 줄에 하나의 연산만 작성할 것인지...

이에 따라 프로그램 코드의 가독성은 높아지기도 낮아지기도 한다.

```
print(str(int(str(3+8)*3)*9)*2)
```

```
99999999999
```

```
x = 3 + 8 # 11
y = str(x) * 3 # '111111'
z = int(y) * 9 # 999999
k = str(z) * 2 # '99999999999'
```

```
print(k)
```

```
99999999999
```

윗 셀의 코드와 아래 셀의 코드는 정확하게 같은 동작을 한다.

다만...

- 윗 셀의 코드는 어떤 연산을 하는 것인지 이해하기가 매우 어렵지만,
- 아래 셀의 코드는 단계별로 따라가기가 쉽다.

물론 때로는 아래 셀과 같이 작성한 코드가 오히려 가독성을 해치기도 한다.
과유불급(過猶不及). 뭐든지 적당한게 좋겠죠?

문자열형

String Type

Python

0	1	2	3	4	5
-6	-5	-4	-3	-2	-1





● 문자열(strings)이란?

● 불변자료형(immutable)

- 내용을 변경할 수 없고 문자의 순서를 바꿀 수 없다
- 시작과 끝을 따옴표로 묶어서 만든 순서가 있는 유니코드 글자(부호)의 배열
 - 시퀀스형(sequence type) 자료구조 중 하나('순서형'이라고도 부른다)
- 문자열 리터럴을 생성하기 위해서는 **작은 따옴표 (' ... ')** 또는 **큰 따옴표 (" ... ")** 사용





예시 : 문자열

'안녕 파이썬'

"안녕 파이썬"

```
print('안녕 파이썬')
```

```
print("안녕 파이썬")
```

'안녕 파이썬'

안녕 파이썬



● 따옴표 안에 따옴표를 표현할 경우

- 작은 따옴표가 들어 있는 경우는 큰 따옴표로 문자열 생성
- 큰 따옴표가 들어 있는 경우는 작은 따옴표로 문자열 생성
- 이스케이프 문자인 역슬래시(\)를 통해 따옴표를 문자열 리터럴을 생성하는 특수문자가 아닌 일반 문자열로 표현 가능

```
print('문자열 안에 '작은 따옴표'를 사용하려면 어떻게 하나요?')
```

```
print("문자열 안에 '작은 따옴표'를 사용하려면 큰 따옴표로 묶는다. ")
```

```
print('문자열 안에 "큰 따옴표"를 사용하려면 작은 따옴표로 묶는다. ')
```

```
print('문자열 안에 같은 \'따옴표\'를 사용하려면 이스케이프 문자를 사용한다. ')
```

```
print("문자열 안에 같은 \"따옴표\"를 사용하려면 이스케이프 문자를 사용한다. ")
```



● 따옴표 3개(triple quotes)

- 작은 따옴표 3개('' '' ... '') 또는 큰 따옴표 3개("''' ... ''''")
- 따옴표 안에 따옴표를 넣을 때 일일이 역슬래시(\)를 넣어주지 않아도 된다
- 문자열 리터럴을 여러 줄에 나눠 입력할 때 편리하다

```
'''안녕 파이썬'''  
  
"""안녕 파이썬"""
```

```
print('''안녕 파이썬''')  
  
print("""안녕 파이썬""")
```

```
'안녕  
파이썬'  
  
File "<stdin>", line 1  
    '안녕  
    ^  
SyntaxError: EOL while scanning string literal
```

```
print('''문자열 안에 '작은 따옴표'나 "큰 따옴표"를 사용할 수 있다.''')  
  
print('''문자열 안에 '작은 따옴표'나  
"큰 따옴표"를 사용할 수도 있고  
여러 줄을 사용할 때도 편리하다.''')
```



이스케이프 문자

● 이스케이프(escape) 문자(탈출 문자) : 역슬래시(\)

- 파이썬 문자열에서 이스케이프 문자인 역슬래시(\)는 역슬래시 바로 다음에 따라오는 문자를 원래 의미가 아닌 특별한 의미로 처리해야 할 때 사용하는 문자
 - \n : 문자 ‘n’으로 처리하지 않고 ‘새줄바꿈’(newline)으로 처리
 - 새줄바꿈 문자 : 한 줄의 끝남을 표시하는 문자로 ‘개행 문자’ 또는 ‘EOL(end-of-line)’과 같은 뜻
 - \t : 문자 ‘t’으로 처리하지 않고 ‘탭(tab)’ 문자로 처리
 - \' : 작은 따옴표를 파이썬 문자열 리터럴을 생성하는 특수 문자로 사용하지 않고 일반적으로 알려진 작은 따옴표 문자로 처리.
 - \": 큰 따옴표를 파이썬 문자열 리터럴을 생성하는 특수 문자로 사용하지 않고 일반적으로 알려진 큰 따옴표 문자로 처리
 - \\" : 화이트스페이스인 새줄바꿈 문자로 처리하지 말고 일반문자인 ‘\n’로 처리

```
print('\\n')
```

```
print('\\n')
```

\n



- print() 함수를 한번만 사용해서 다음 문장을 아래 3가지 방법으로 출력하시오

- 삼중 따옴표로 문자열 생성
- 작은 따옴표로 문자열 생성
- 큰 따옴표로 문자열 생성

공백문자 한 개 삽입
탭 한 개 삽입

President Barack Obama said,
"Don't just play on your phone,
program it."



문자열 내에서 역슬래시(\)가 이스케이프 문자가 아니라
원래 의미인 역슬래시 문자라면?

더욱이 문자열 내에 역슬래시 문자를 많이 사용해야 한다면?



로스트링을 사용하면 이 문제를 해결할 수 있다

```
print('C:\home\nations\name\nationality\nature')
```

```
print('C:\home\\nations\\name\\nationality\\nature')
```

```
print(r'C:\home\nations\name\nationality\nature')
```

로스트링(raw string)이란?

- 문자열을 나타내는 첫 따옴표 앞에 **r**을 붙여
- 해당 문자열 내부의 모든 문자는 그대로 인식하게 한다
- 따라서, 걱정없이 이스케이프 문자를 마음껏 사용할 수 있다



● 화이트스페이스 whitespace 란?

- 화면상으로는 표시되지 않지만 컴퓨터 모니터 화면이나 프린터 등과 같은 출력 장치를 제어하는 문자
- e.g., 공백문자(' '), 새줄바꿈('\n'), 탭('\t'), 캐리지 리턴('\r'), 수직탭('\v' 또는 '\x0b'), 페이지 넘기기('\f' 또는 '\x0c') 등

● 화이트스페이스와 `print()` 함수

```
y = '\t일\n월\n화\n수\n목\n금\n\t토'
```

```
y
```

```
print(y)
```



줄 연결(line continuation)

- 지나치게 긴 문장을 여러 줄에 나눠 작성하는 것이 가능
- 문장이 너무 길면 화면에서 스크롤링 없이 보기 불편할 뿐만 아니라 출력 시에도 매우 번거롭다
 - <PEP 8 파이썬 코드 스타일 가이드>은 한 줄에 최대 79자로 제한할 것을 권고
- 이처럼 긴 문장을 작성해야 할 경우 한 줄에 모두 작성하는 것보다 여러 줄에 나누어 작성하는 것이 좋다

문장을 여러 줄에 나누어 작성하는 방법

- 소괄호(())
- 역슬래시(\) : 새줄바꿈 방지(탈출)

```
x = ('소괄호를 사용하면 여러 줄로 작성한 문자열을 간편하게 합할 수 있다.  
'표현식으로 이루어지기 때문에 깔끔하게 보인다.')
```

```
x
```

```
y = '이스케이프 문자인 역슬래시를 사용해서 여러 줄로 작성한 문자열을 '\  
'합할 수도 있지만 PEP 8 파이썬 코드 스타일 가이드에서 추천하지 않는다.'
```

```
y
```

```
# not recommended  
1 + 2 \  
* 3
```

```
7
```

```
# better  
(1 + 2  
* 3)
```

```
7
```



문자열 인덱스

● 문자열은 시퀀스형(sequence) 자료 구조

- ‘문자열’은 각 문자의 위치가 정해져 있는 즉, 순서가 있는 문자의 배열
- ‘문자’는 말 그대로 한 개의 문자를 가리킴
 - 파이썬에는 **char** 자료형이 존재하지 않음
- 문자가 순서를 가지고 정해진 위치에 들어가게 되면 문자의 배열인 문자열이 생성됨

● 문자열 인덱스(index)

- 인덱스** : 배열 안에서 각 값의 위치

H	i		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

안	녕		파	이	썬
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

- 첫 문자의 인덱스 : [0]

- 마지막 문자의 인덱스 : [$n - 1$] 또는 [-1]

n = 문자열의 전체 크기

인덱스 [-1]을 사용하면 항상 마지막 문자를 찾을 수 있으므로 매우 유용

● IndexError 예외

- 문자열 범위를 벗어난 인덱스나 빈 문자열에 접근하려고 하면 **IndexError** 예외가 발생



예시 : 문자열에서 개별 문자 가져오기

s = '안녕 파이썬'

s[0]

s[2]

s[4]

s[6]

s[-1]

s[-3]

s[-6]

s[-7]



문자 한 개 이상 여러 개를 한꺼번에 가져올 수는 없을까?

예를 들어, '안녕 파이썬'에서 '파이썬'을 한 번에 추출할 수는 없을까?



- **분할(slicing) 연산자 : [:], [::]**

- 문자열의 전부 또는 일부를 추출

문자열[시작 : 끝]

- 시작과 끝은 생략 가능
- 시작을 생략하면 기본 값인 0에서 시작
- 끝을 생략하면 기본 값인 `len(문자열)`까지 가져온다
- 시작과 끝을 모두 생략하면 모든 대상을 가져온다
- 문자열[:]는 문자열[0 : len(문자열)]과 같다

문자열[시작 : 끝 : 폭]

- 시작과 끝은 생략 가능
- 시작을 생략하면 기본 값인 0에서 시작
- 끝을 생략하면 기본 값인 `len(문자열)`까지 가져옴
- 폭이 음수라면 시작은 -1에서 시작(뒤에서부터 시작하는 뜻)

★ `len(문자열)` : 문자열의 크기를 반환

문법	설명
<code>seq[시작]</code>	<ul style="list-style-type: none"> ▶ <code>seq</code> 의 시작 번째에 있는 대상을 추출 <ul style="list-style-type: none"> - <code>seq</code> := 시퀀스형(sequence) 자료(e.g., 문자열, 리스트, 튜플) - 시작, 끝, 폭 := 정수 또는 정수 값을 담은 변수
<code>seq[시작 : 끝]</code>	<ul style="list-style-type: none"> ▶ 시작 번째에 있는 대상부터 끝 번째 앞에 있는 대상까지 추출
<code>seq[시작 : 끝 : 폭]</code>	<ul style="list-style-type: none"> ▶ 시작 번째에 있는 대상부터 끝 번째 앞에 있는 대상까지 추출 하는데 모든 대상을 추출하지 않고 폭 만큼 건너뛰어가며 추출



예시 : 문자열 분할

```
s = 'abcde 12345'
```

```
len(s)
```

```
s[::]
```

```
s[0:len(s)]
```

```
s[2:8]
```

```
s[2::]
```

```
s[-7::]
```

```
len(s[1:3])
```

```
s[::3]
```

```
s[: -4 : 3]
```

```
s[:: -2]
```

```
s[ : 4 : -2 ]
```

```
s[2:4:-2]
```



범위를 벗어난 인덱스에 접근

- 예외가 발생
- 하지만... 문자열 분할(slicing)을 사용할 경우는 범위를 벗어난 인덱스에 접근해도 오류가 발생하지 않음

```
s = '초급자를 위한 파이썬'  
len(s)  
s[11]
```

```
s[11:  
s[10]  
s[8:11]  
s[8:99]
```



문자열 분할 연산으로 어떤 것을 할 수 있을까?

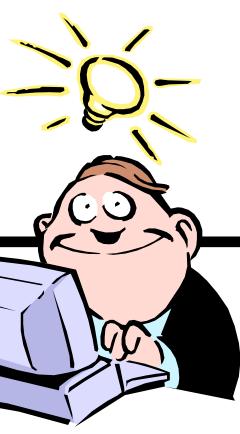


분할 연산자 `[::]`로 문자열을 분할해서 합치면 간단한 암호를 만들 수 있다

```
text = 'Hello' + 'Python'  
x = text[1::2]  
y = text[::-2]  
z = x + y  
  
print(z)  
  
elPtoHloyhn
```

코드 설명

- ❶ 두 문자열을 합해서 'HelloPython'을 만든 후
- ❷ 합친 문자열 `text`에서
 - ❸ 짝수번째 글자를 먼저 추출해서 변수 `x`에 담고
 - ❹ 홀수번째 글자를 추출해서 변수 `y`에 담은 후
- ❺ 이 둘을 합쳐서 변수 `z`에 담는다.



앞서 배운 암호 작성술로 "무궁화꽃이피었습니다"를 암호화 해보자.



● 문자열 일부 교체/추가/삭제

- 문자열의 일부분을 다른 문자열로 교체하거나 문자열 안에 추가로 문자열을 집어 넣을 경우 **예외**가 발생
 - 그 이유는 문자열이 **불변자료형(immutable)**이기 때문
 - 즉, 생성한 후 값을 변경할 수 없기 때문에 수정이 불가능

```
s = 'Python'  
s[1] = 'i' # Python으로 변경
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: 'str' object does not support item assignment
```

- 그러나... 문자열 **분할과 결합(slicing with concatenation)** 또는 **str.replace()** 메소드를 이용해 같은 변수에 재할당하면 이 문제를 해결할 수 있음
- replace()**를 함수라 하지 않고 메소드(method)라 부르는 이유는 **replace()**가 문자열 클래스에 속해 있는 함수이기 때문
 - 함수는 특정 클래스에 속해있지 않음
 - ★ 메소드에 관해서는 마지막 장 클래스에서 다룸



예시 : 문자열 수정

```
s = '초급자를 위한 파이썬'
```

```
s[0] = '중' # 첫 문자를 '중'으로 대체한다.
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
s = '중' + s[1:]
```

```
s
```

```
s = s[:8] + '재미있는 ' + s[8:]
```

```
s
```



문자열을 분할하고 결합하기

- 사용자의 입력을 받은 후, 입력받은 문자열의 첫 글자와 마지막 글자의 위치를 바꾸어 출력
- 입력하는 문자열의 길이는 2 이상이라 가정
- 그 외 다양한 길이의 입력에 대해 모두 정상적으로 작동해야 한다

실행 결과 예시

```
단어나 문장을 입력하세요....: 동해물과 백두산이  
이해물과 백두산동  
  
단어나 문장을 입력하세요....: 사과  
과사  
  
단어나 문장을 입력하세요....: What a wonderful world  
dhat a wonderful worlW
```



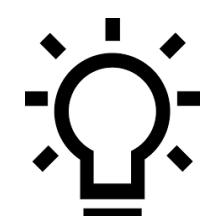
문자열 분할 : **split()** 메소드

- 문자열 **.split(*sep=None, maxsplit=-1*)**
.rsplit(*sep=None, maxsplit=-1*)

- split() 메소드는 *sep*를 기준으로 문자열을 왼쪽부터 나눈 후 리스트 형태로 반환
- rsplit() 메소드는 *sep*를 기준으로 문자열을 오른쪽부터 나눈 후 리스트 형태로 반환

- sep*는 구분자로 문자열을 분할할 때의 기준
 - 구분자 *sep*를 생략하면 기본값인 화이트스페이스 문자 기준으로 분할
 - 구분자 *sep*는 반환하는 리스트에 포함되지 않는다

- maxsplit*는 최대 분할 값으로 이 값만큼만 구분자 *sep*로 분할
 - 문자열에서 구분자 *sep*의 개수가 최대 분할 값 *maxsplit*보다 작다면, 구분자 *sep*를 기준으로 모두 분할
 - 최대 분할 값 *maxsplit*를 생략하거나 기본값인 -1이 주어지면 구분자 *sep*만큼 모두 분할



<문자열>.splitlines()

- 각 줄의 마지막에 새줄바꿈 등 화이트스페이스 없이 줄 단위로 나눈 후 리스트 형식으로 반환

*참고: 리스트(list)

- 복합 자료형, 순회형
- 한 개 이상의 값(원소, 객체)을 가질 수 있다
- 대괄호([])를 사용하여 선언
- 각 값(원소, 객체)들은 콤마(,)를 사용하여 구분
- 예) my_list [a, b, c, d]



문자열 분할 : `split()` 메소드

```
kor = '파이썬을 배우면서 파이썬을 즐기자!!!'
```

```
# 기본값은 화이트스페이스 문자 기준으로 모두 분할해서 리스트로 반환한다.  
kor.split()
```

```
# '을' 기준으로 모두 분할한다.  
kor.split('을') # split(sep='을')와 같다.
```

```
# '을' 기준으로 오른쪽부터 한 번만 분할한다.  
kor.rsplit('을', 1) # rsplit(sep='을', maxsplit=1)와 같다.
```

```
# 화이트스페이스 문자 기준으로 왼쪽부터 두 번만 분할한다.  
kor.split(maxsplit=2)
```

```
# 화이트스페이스 문자 기준으로 오른쪽부터 한 번만 분할한다.  
kor.rsplit(maxsplit=1)
```

```
eng = 'Introduction to Python'
```

```
# 화이트스페이스 문자 기준으로 오른쪽부터 모두 분할한다.  
eng.rsplit()
```

```
# 't' 기준으로 왼쪽부터 두 번만 분할한다.  
eng.split(sep='t', maxsplit=2)
```

```
# 'o'를 기준으로 오른쪽부터 한 번만 분할한다.  
eng.rsplit('o', maxsplit=1)
```

```
# 'o'를 기준으로 오른쪽부터 모두 분할한다.  
eng.rsplit('o') # eng.split('o')와 결과는 같다.
```



문자열 분할 : `split()` 메소드

주의 1

- 지정한 구분자 `sep` 가 문자열의 맨 끝에 있으면 반환하는 리스트의 마지막에 빈 문자열을 포함한다
- `split()` 는 구분자를 기준으로 양쪽을 반환하는데, 구분자의 오른쪽에 아무 것도 없기 때문에 오른쪽 끝은 빈 문자열로 반환하기 때문이다

```
'Introduction to Python'.split('\n')
```

```
'두 줄입니다\n'.split('\n')
```

주의 2

- 구분자 `sep` 를 지정하지 않았다면, 연속적으로 있는 화이트스페이스를 하나의 구분자로 취급한다
- 따라서 빈 문자열이나 화이트스페이스로만 이루어져 있는 문자열은 빈 리스트를 반환

```
'\t \n \n\t \t '.split()
```

```
'.split()
```

```
'.split()
```

```
' 1 2 3 '.split()
```

```
' \t 1 \n 2\n\t \t3 '.split()
```



문자열 결합 : `join()` 메소드

결합문자열 `.join(순회형)`

- 여러 문자열을 한꺼번에 연결할 때 사용하며, 순회형 자료를 결합문자열로 결합해서 문자열 한 개로 반환하는 함수
- 결합문자열은 문자열을 결합할 때 사용하는 문자열
- 순회형(`iterable`)은 순회 또는 반복이 가능한 자료형으로 문자열, 리스트, 튜플, 딕셔너리, 세트 등이 있다
- 순회형의 문자열들을 결합문자열로 결합해서 하나의 문자열로 반환하기 때문에 순회형의 모든 객체들은 반드시 문자열이어야 한다
- 많은 수의 문자열을 한번에 연결하는 것이 가능

```
instruments = ['drum', 'guitar', 'bass', 'keyboard']
```

```
' '.join(instruments)
```

```
' '.join(instruments)
```

```
', '.join(instruments)
```

```
' + ' .join(reversed(instruments))
```

```
'+'.join(['파이썬', 5])
```



문자열 삭제 : `strip()` 메소드

- 문자열 `.strip([삭제할-문자-세트])`
- `.lstrip([삭제할-문자-세트])`
- `.rstrip([삭제할-문자-세트])`

- 이 세 메소드는 문자열에서 삭제할-문자-세트를 제거한 문자열을 반환하는 함수
 - `strip()`은 삭제할-문자-세트를 문자열의 양쪽 끝부터 삭제한 문자열을 반환
 - `lstrip()`은 삭제할-문자-세트를 문자열의 왼쪽 끝부터 삭제한 문자열을 반환
 - `rstrip()`은 삭제할-문자-세트를 문자열의 오른쪽 끝부터 삭제한 문자열을 반환
- 삭제할-문자-세트는 삭제할 문자들의 집합으로 문자열의 끝부터 삭제할-문자-세트에 포함되지 않은 문자에 도달할 때까지 계속 삭제
- 이때 삭제할-문자-세트에 포함된 모든 문자를 문자열에서 삭제하기 때문에 삭제할-문자-세트에 포함된 문자의 순서는 중요하지 않다
- 만약 삭제할-문자-세트를 지정하지 않으면, 기본값인 화이트스페이스 문자를 문자열에서 삭제

```
s = '\t 초급자를\n위한 파이썬!!!'
```

```
s.strip()
```

```
s.lstrip()
```

```
s.rstrip()
```

```
s.strip('!\t\n')
```

```
s.strip('! \t')
```

```
'Right way to Python'.strip('goPniR')
```



문자열 교체 : `replace()` 메소드

문자열 `.replace(기존-부분문자열, 새로운-부분문자열[, 교체횟수])`

- 문자열 중 기존-부분문자열을 모두 새로운-부분문자열로 교체한 새 문자열을 반환
- 선택 사항인 교체횟수(count)를 지정하면, 그 교체횟수만큼만 교체한 새 문자열을 반환
- 부분문자열이란 문자열의 일부 또는 전부를 뜻한다
- 새로운-부분문자열을 빈 문자열('')로 설정하면, 원하는 문자를 삭제하는 효과를 가질 수 있다

'초급자를 위한 파이썬' `.replace('초', '중')`

'I love Python programming'.`replace('o', '0', 2)`

'초급자를 위한 파이썬' `.replace('를 위한', '')`



문자열 교체 : 영어 관련 메소드

문자열.title()

- 문자열 각 단어의 첫 문자를 대문자로 반환

```
s = 'Right Way to 파이썬 programming'
```

```
s.title()
```

문자열.capitalize()

- 문자열의 첫 문자만 대문자로 반환

```
s.capitalize()
```

문자열.swapcase()

- 문자열의 대문자는 소문자로 소문자는 대문자로 반환

```
'파이썬 right way to programming'.capitalize()
```

문자열.lower()

- 문자열의 모든 문자를 소문자로 반환

```
s.swapcase()
```

문자열.upper()

- 문자열의 모든 문자를 대문자로 반환

```
s.lower()
```

```
s.upper()
```



문자열의 삭제/교체 메소드

- 처리한 문자열을 메소드의 결과로 반환한다
- 메소드를 호출한 문자열은 변하지 않는다

```
s = '바로 쓰는 파이썬'  
t = s.replace('파이썬', 'Python')  
  
print(s)  
  
print(t)
```



교체/분할/결합 메소드를 사용하여 문자열 처리하기

- 다음은 영화 리뷰 사이트 중 하나인 IMDb(<https://www.imdb.com>)에 올라온 영화 리뷰 중 일부

I never watched movie two times in cinema. This is the only one I did. I watched on CGV cinema, had my tears, and decided to try the IMAX experience.

- 리뷰의 분석을 위하여 사용된 단어들의 목록을 추려내려고 한다

- 문자열 메소드들을 활용하여 리뷰에 사용한 단어들을 콤마로 분리한 형태로 출력



힌트

- 단어가 아닌 문장 부호(' ','.')는 모두 제거
- 모든 영문자를 소문자로 변환
- 단어들을 분할하여 리스트로 변환
- 리스트의 값들을 문자열 결합

- 실행 결과 예시

```
i,never,watched,movie,two,times,in,cinema,this,is,the,only,one,i,did,i,watched,  
on,cgv,cinema,had,my,tears,and,decided,to,try,the,imax,experience
```



<https://github.com/snu-python/pythonbook/blob/master/Appendix3%20String%20Methods.pdf>

자세한 내용은
<바로 쓰는 파이썬: 기초 편>의
<부록 3 - 유용한 문자열 메소드>를
참조하기 바란다



출력 결과는 간단하지만 출력하는 방법은 복잡하다

```
x = 3
y = 5
z = 19

print(str(x) + ' x ' + str(y) + ' = ' + str(z) + ' (' + str(x * y == z) + ')') # 문자열 결합
print(x, 'x', y, '=', z, ' (' + str(x * y == z) + ')') # 개별 객체

3 x 5 = 19 (False)
3 x 5 = 19 (False)
```

하지만...

string.format() 메소드를 이용하면 출력을 편하게 할 수 있다

```
print('{} x {} = {} ({})'.format(x, y, z, x * y == z))
3 x 5 = 19 (False)
```



다음 장에서 배울 **for** 순환문에서도 **format()**을 사용하면 출력을 간편하게 할 수 있다

```
for i in range(1, 10):
    print('{} x {} = {}'.format(i, 5, i * 5))
```

```
1 x 5 = 5
2 x 5 = 10
3 x 5 = 15
4 x 5 = 20
5 x 5 = 25
6 x 5 = 30
7 x 5 = 35
8 x 5 = 40
9 x 5 = 45
```



문자열 서식 : **format()** 메소드

● 문자열 **.format()** 메소드

- 이 함수를 활용해 문자열 생성을 매우 유연하고 다양하게 설정하는 것이 가능

● 언제 사용?

- 문자열 내의 특정한 값을 바꿔야 할 경우에 유용
- 대체필드(replacement field)를 사용해서 특정한 값을 바꿀 수 있다
 - 대체필드는 중괄호({ })로 표시

● 어떻게 사용?

- 문자열 내부에 위치한 대체필드(replacement field)를 **format()**의 전달인자로 대체하여 새로운 문자열을 반환
- 대체필드를 설정할 때는 중괄호({ })를 넣고 그 안에 ‘필드명’을 입력
 - ‘필드명’은 전달하려는 위치 전달인자(positional argument)의 인덱스나 키워드 전달인자(keyword argument)의 이름 중 하나를 선택
 - 위치 전달인자 : 대체필드({ }) 안의 ‘필드명’에 위치 번호(0, 1, ...)을 입력해도 되고 생략해도 된다
 - 키워드 전달인자 : 대체필드({ }) 안의 ‘필드명’이 들어갈 자리에 ‘키워드’를 입력
 - 본 수업은 기초 파이썬 강좌이기 때문에 간단한 위치 전달인자만 다룬다



예시 : format() 메소드의 위치 전달인자 설정

```
'1 + 2 = {0}'.format(1 + 2)
```

```
name = '라이언'  
'저의 이름은 {0}입니다.'.format(name)
```

```
'{0} * {1} = {2}입니다.'.format(3, 5, 3 * 5)
```

```
'{1} * {0} = {2}입니다.'.format(3, 5, 3 * 5)
```

```
for i in range(1, 10): # i의 값이 1부터 9까지 주어짐  
    print('{0} * {1} = {2}'.format(5, i, 5 * i))
```

```
fruits = ['사과', '딸기', '바나나', '배', '블루베리']  
for fruit in fruits:  
    print('나는 {0}를 좋아해요.'.format(fruit))
```

예시 : **format()** 메소드의 위치 전달인자 설정(파이썬 3.1 버전 이후)

```
'1 + 2 = {}'.format(1 + 2)
```

```
name = '라이언'  
'저의 이름은 {}입니다.'.format(name)
```

```
'{} * {} = {}입니다.'.format(3, 5, 3 * 5)
```

```
for i in range(1, 10): # i의 값이 1부터 9까지 주어짐  
    print('{} * {} = {}'.format(5, i, 5 * i))
```

```
fruits = ['사과', '딸기', '바나나', '배', '블루베리']  
for fruit in fruits:  
    print('나는 {}를 좋아해요.'.format(fruit))
```



예시 : format() 메소드의 숫자 서식설정

```
# 쉼표로 정수를 그룹화해서 출력  
'{0:,}'.format(123456789)
```

```
# 최소 출력 길이를 15로 해서 정수를 우측 정렬 후 빈 칸은 '*'로 채우고 쉼표로 그룹화해서 출력  
'{0:*>15,}'.format(123456789)
```

```
# 실수를 소수점 두 번째 자리로 반올림해서 출력  
'{:.2f}'.format(123456789.12534)
```

```
# 실수를 소수점 세 번째 자리로 반올림해서 출력  
'{:.3f}'.format(123456789.12534)
```

```
# 쉼표로 실수를 그룹화하고 소수점 두 번째 자리로 반올림해서 출력  
'{:, .2f}'.format(123456789.12534)
```

```
# 쉼표로 실수를 그룹화하고 소수점 두 번째 자리로 반올림해서 출력  
'{:>15,.2f}'.format(123456789.12534)
```



format() 메소드 참고 자료

<https://github.com/snu-python/pythonbook/blob/master/Appendix4%20Format%20Specification.pdf>

자세한 내용은
<바로 쓰는 파이썬: 기초 편>의
<부록 4 - 문자열 서식 설정>을
참조하기 바란다



문자열 서식 : f-string

● format() 메소드의 문제점

- 표현식이 format() 메소드의 전달인자로 오기 때문에 실제 대체 필드 { } 와 동떨어져 있어 직관적이지 않다

● f-string

- f 를 접두사로 가지는 문자열 리터럴로서 문자열 안의 표현식은 중괄호로 묶여져 있다

- 문자열 앞에 f 를 접두사로 사용

- 파이썬 3.6 이후 버전부터 사용이 가능

● f-string 장점

- 최소한의 문법을 사용하여 문자열 리터럴 안에 표현식을 포함시킬 수 있다
- 따라서 %-formatting 이나 format() 메소드보다 간결하게 표현할 수 있다

```
x = 3  
y = 5  
z = 19
```

```
print('{} x {} = {} ({})'.format(x, y, z, x * y == z))
```

```
3 x 5 = 19 (False)
```

```
print(f'{x} x {y} = {z} ({x * y == z})')
```

```
3 x 5 = 19 (False)
```



예시 : f-string

```
import datetime

anniversary = datetime.date(1988, 7, 1)

# format() 메소드보다 표현식이 간결하다.
# [참고] %A - 요일, %B - 월, %d - 일, %Y - 년도
print('My anniversary is {}.'.format(anniversary.strftime('%A, %B %d, %Y'))))
print(f'My anniversary is {anniversary:%A, %B %d, %Y}.')
```

My anniversary is Friday, July 01, 1988.

My anniversary is Friday, July 01, 1988.

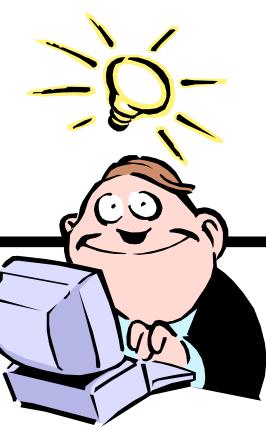


<https://github.com/snu-python/pythonbook/blob/master/Appendix4%20Format%20Specification.pdf>

자세한 내용은
<바로 쓰는 파이썬: 기초 편>의
<부록 4 - 문자열 서식 설정>을
참조하기 바란다



Lab : **format()** / **f-string** 활용



● **format()** 이나 **f-string** 활용하기

- 직원 세 명의 성명, 나이, 월급에 대한 정보를 입력받아, 표 형태로 출력
- 직원 정보 입력에서 각 필드는 콤마(,)로 구분되며, 이름 필드에는 영문만 입력한다고 가정
- 표에서 각 열의 너비는 세로줄을 제외하고 15로 설정 (가로줄 '='의 길이는 48로 설정)

💡 힌트

- split()** 메소드의 결과로 반환하는 리스트는, 문자열과 같은 방법으로 인덱싱이 가능
- 예) `person1 = 'Muzi,25,2500000'.split(',')`
 - person1[0]의 값 : 'Muzi',
 - person1[2]의 값 : '2500000'

● 실행 결과 예시

```
직원1에 대한 정보(성명, 나이, 월급)를 입력하세요: Muzi,25,2500000
직원2에 대한 정보(성명, 나이, 월급)를 입력하세요: Ryan,47,8000000
직원3에 대한 정보(성명, 나이, 월급)를 입력하세요: Neo,35,4000000
NAME          | AGE           | SALARY        |
=====
Muzi          | 25            | 2,500,000    |
Ryan          | 47            | 8,000,000    |
Neo           | 35            | 4,000,000    |
```