02

파이썬 프로그래밍의 기초, 자료형

02-1 숫자형

숫자형(Number)이란?

- 숫자 형태로 이루어진 자료형

항목	파이썬 사용 예
정수	123, -345, 0
실수	123.45, -1234.5, 3.4e10
8진수	0o34, 0o25
16진수	0x2A, 0xFF

02-1 숫자형

숫자형 사용법

정수형(Integer)

- 정수를 뜻하는 자료형

```
>>> a = 123
>>> a = -178
>>> a = 0
```

실수형(Floating-point)

- 소수점이 포함된 숫자

```
>>> a = 1.2
>>> a = -3.45
```

```
>>> a = 4.24E10
>>> a = 4.24e-10
```

※ 컴퓨터식 지수 표현 방식

8진수(Octal)

- 숫자 0 + 알파벳 소문자 o 또는 대문자 O

```
>>> a = 0o177
```

16진수(Hexadecimal)

- 숫자 0 + 알파벳 소문자 x

```
>>> a = 0x8ff
>>> b = 0xABC
```

02-1 숫자형

▪ 숫자형 활용하기 위한 연산자

▪ 사칙연산

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a * b
12
>>> a / b
0.75
```

▪ // 연산자 : 나눗셈 후 몫 반환

```
>>> 7 // 4
1
```

▪ ** 연산자 : 제곱

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

▪ % 연산자 : 나눗셈 후 나머지 반환

```
>>> 7 % 3
1
>>> 3 % 7
3
```

02-2 문자열 자료형

▪ 문자열(String)이란?

- 문자, 단어 등으로 구성된 문자들의 집합

```
"Life is too short, You need Python"
"a"
"123"
```

02-2 문자열 자료형

▪ 문자열 사용법

▪ 문자열 만드는 방법

1. 큰따옴표(")

```
"Hello World"
```

2. 작은따옴표(')

```
'Python is fun'
```

3. 큰따옴표 3개(""")

```
"""Life is too short, You need python"""
```

4. 작은따옴표 3개(''')

```
'''Life is too short, You need python'''
```

02-2 문자열 자료형

▪ 문자열 사용법

▪ 문자열 안에 작은따옴표나 큰따옴표를 포함시키기

1. 작은따옴표(')
   - 큰따옴표(")로 둘러싸기

```
>>> food = "Python's favorite food is perl"
```

2. 큰따옴표(")
   - 작은따옴표(')로 둘러싸기

```
>>> say = '"Python is very easy." he says.'
```

3. 백슬래시(₩) 활용하기
   - 백슬래시(₩) 뒤의 작은따옴표(')나 큰따옴표(")는 문자열을 둘러싸는 기호의 의미가 아니라 문자 ('), (") 그 자체를 의미

```
>>> food = 'Python\'s favorite food is perl'
>>> say = "\"Python is very easy.\" he says."
```

02-2 문자열 자료형

▪ 문자열 사용법

▪ 여러 줄인 문자열을 변수에 대입하고 싶을 때

1. 이스케이프 코드 '₩n' 삽입

```
>>> multiline = "Life is too short\nYou need python"
```

2. 작은따옴표 3개(''')

```
>>> multiline = '''
... Life is too short
... You need python
... '''
```

3. 큰따옴표 3개(""")

```
>>> multiline = """
... Life is too short
... You need python
... """
```

02-2 문자열 자료형

▪ 문자열 연산하기

1. 문자열 더해서 연결하기(Concatenation)

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

2. 문자열 곱하기

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

3. 문자열 길이 구하기

- 파이썬 기본 내장 함수 len( )

```
>>> a = "Life is too short"
>>> len(a)
17
```

02-2 문자열 자료형

▪ 문자열 인덱싱

▪ 인덱싱(Indexing)

- '가리킨다'는 의미
- 파이썬은 0부터 숫자를 셈

- a[번호]
  - 문자열 안의 특정 값 뽑아냄
  - 마이너스(-)
    - 문자열 뒤부터 셈

L	i	f	e		i	s		t	o	o		s	h	o	r	t	,		Y	o	u		n	e	e	d		P	y	t	h	o	n
0										1										2										3			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3

```
>>> a = "Life is too short, You need Python"
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
```

```
a[0]: 'L', a[1]: 'i', a[2]: 'f', a[3]: 'e', a[4]: ' ', …
```

02-2 문자열 자료형

▪ 문자열 슬라이싱

▪ 슬라이싱(Slicing)

- '잘라낸다'는 의미
- a[시작 번호:끝 번호]
  - 시작 번호부터 끝 번호까지의 문자를 뽑아냄
  - 끝 번호에 해당하는 것은 포함하지 않음

L	i	f	e		i	s		t	o	o		s	h	o	r	t	,		Y	o	u		n	e	e	d		P	y	t	h	o	n
0										1										2										3			
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3

```
>>> a = "Life is too short, You need Python"
>>> a[0:4]
'Life'
```

```
>>> a = "20010331Rainy"
>>> date = a[:8]
>>> weather = a[8:]
>>> date
'20010331'
>>> weather
'Rainy'
```

02-2 문자열 자료형

▪ 문자열 포매팅

▪ 포매팅(Formatting)

1. 숫자 바로 대입
   - 문자열 포맷 코드 **%d**

```
>>> "I eat %d apples." % 3
'I eat 3 apples.'
```

```
>>> number = 3
>>> "I eat %d apples." % number
'I eat 3 apples.'
```

2. 문자열 바로 대입
   - 문자열 포맷 코드 **%s**

```
>>> "I eat %s apples." % "five"
'I eat five apples.'
```

02-2 문자열 자료형

▪ 문자열 포매팅

▪ 문자열 포맷 코드

코드	설명
%s	문자열(String)
%c	문자 1개(Character)
%d	정수(Integer)
%f	부동 소수(Floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 '%' 자체)

```
>>> number = 10
>>> day = "three"
>>> "I ate %d apples. so I was sick for %s days." % (number, day)
'I ate 10 apples. so I was sick for three days.'
```

```
>>> "I have %s apples" % 3
'I have 3 apples'
>>> "rate is %s" % 3.234
'rate is 3.234'
```

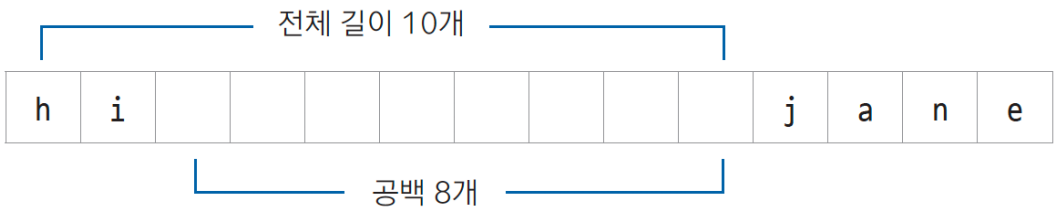
02-2 문자열 자료형

▪ 문자열 포매팅

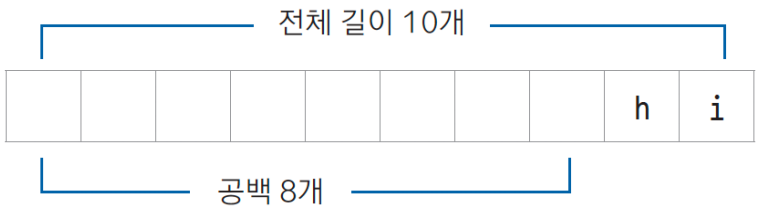
▪ 문자열 포맷 코드 활용법

1. 정렬과 공백
   - %s를 숫자와 함께 사용하면, 공백과 정렬 표현 가능

```
>>> "%10s" % "hi"
'        hi' ◀── hi가 오른쪽 정렬됨
```




```
>>> "%-10sjane" % 'hi'
'hi        jane' ◀── hi가 왼쪽 정렬됨
```



02-2 문자열 자료형

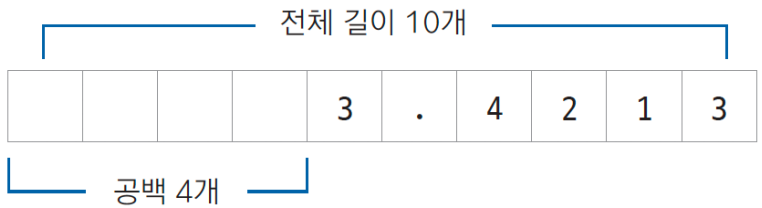
▪ 문자열 포매팅

▪ 문자열 포맷 코드 활용법

2. 소수점 표현하기
   - %f를 숫자와 함께 사용하면, 소수점 뒤에 나올 숫자의 개수 조절 및 정렬 가능

```
>>> "%0.4f" % 3.42134234
'3.4213'
```

```
>>> "%10.4f" % 3.42134234
'    3.4213'
```



02-2 문자열 자료형

▪ 문자열 포매팅

▪ f 문자열 포매팅

- 파이썬 3.6 버전부터 f 문자열 포매팅 기능 제공
- 문자열 앞에 f 접두사를 붙이면, f 문자열 포매팅 기능 사용 가능

```
>>> name = '홍길동'
>>> age = 30
>>> f'나의 이름은 {name}입니다. 나이는 {age}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.'
```

```
>>> age = 30
>>> f'나는 내년이면 {age+1}살이 된다.'
'나는 내년이면 31살이 된다.'
```

02-2 문자열 자료형

▪ 문자열 관련 함수

- 문자열 자료형이 가진 내장 함수

- **count( )**
  - 문자 개수 세는 함수

```
>>> a = "hobby"
>>> a.count('b')
2
```

- **find( )**
  - 찾는 문자열이 처음 나온 위치 반환
  - 없으면 -1 반환

```
>>> a = "Python is the best choice"
>>> a.find('b')
14 ◀── 문자열에서 b가 처음 나온 위치
>>> a.find('k')
-1
```

02-2 문자열 자료형

▪ 문자열 관련 함수

▪ index( )

- find와 마찬가지로,
  찾는 문자열이 처음 나온 위치 반환
- 단, 찾는 문자열이 없으면 오류 발생

```
>>> a = "Life is too short"
>>> a.index('t')
8
>>> a.index('k')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

k가 없기 때문에 오류 발생

▪ join( )

- 문자열 삽입

```
>>> ",".join('abcd')
'a,b,c,d'
```

▪ upper( )

- 소문자를 대문자로 변환

```
>>> a = "hi"
>>> a.upper()
'HI'
```

02-2 문자열 자료형

▪ 문자열 관련 함수

▪ lower( )

- 대문자를 소문자로 변환

```
>>> a = "HI"
>>> a.lower()
'hi'
```

▪ lstrip( )

- 가장 왼쪽에 있는 연속된 공백 삭제

```
>>> a = " hi "
>>> a.lstrip()
'hi '
```

▪ rstrip( )

- 가장 오른쪽에 있는 연속된 공백 삭제

```
>>> a= " hi "
>>> a.rstrip()
' hi'
```

▪ strip( )

- 양쪽에 있는 연속된 공백 삭제

```
>>> a = " hi "
>>> a.strip()
'hi'
```

02-2 문자열 자료형

▪ 문자열 관련 함수

▪ replace( )

- replace(바뀌게 될 문자열, 바꿀 문자열)
- 문자열 안의 특정 값을 다른 값으로 치환

```
>>> a = "Life is too short"
>>> a.replace("Life", "Your leg")
'Your leg is too short'
```

▪ split( )

- 공백 또는 특정 문자열을 구분자로 해서 문자열 분리
  분리된 문자열은 리스트로 반환됨

```
>>> a = "Life is too short"
>>> a.split() ◀── 공백을 기준으로 문자열 나눔
['Life', 'is', 'too', 'short']
>>> b = "a:b:c:d"
>>> b.split(':') ◀── : 기호를 기준으로 문자열 나눔
['a', 'b', 'c', 'd']
```

02-3 리스트 자료형

▪ 리스트(List)란?

- 자료형의 집합을 표현할 수 있는 자료형

```
>>> odd = [1, 3, 5, 7, 9]
```

- 숫자와 문자열만으로 프로그래밍을 하기엔 부족한 점이 많음
  - 예) 1부터 10까지의 숫자 중 홀수 모음인 집합 {1, 3, 5, 7, 9}는 숫자나 문자열로 표현 불가능
  - 리스트로 해결 가능!

02-3 리스트 자료형

▪ 리스트 사용법

- 대괄호([ ])로 감싸고 각 요솟값은 쉼표(,)로 구분

```
리스트명 = [요소1, 요소2, 요소3, …]
```

- 리스트 안에 어떠한 자료형도 포함 가능

```
>>> a = []
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```

02-3 리스트 자료형

▪ 리스트 인덱싱

- 문자열과 같이 인덱싱 적용 가능

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
```

- 파이썬은 숫자를 0부터 세기 때문에 a[0]이 리스트 a의 첫 번째 요소

```
>>> a[0]
1
```

- a[-1]은 리스트 a의 마지막 요솟값

```
>>> a[-1]
3
```

- 요솟값 간의 덧셈

```
>>> a[0] + a[2] ◀── 1 + 3
4
```

02-3 리스트 자료형

▪ 리스트 인덱싱

- 리스트 내에 리스트가 있는 경우

```
>>> a = [1, 2, 3, ['a', 'b', 'c']]
```

- a[-1]은 마지막 요솟값인 리스트 ['a', 'b', 'c'] 반환

```
>>> a[0]
1
>>> a[-1]
['a', 'b', 'c']
```

- 리스트 a에 포함된 ['a', 'b', 'c'] 리스트에서 'a' 값을 인덱싱을 사용해 반환할 방법은?

```
>>> a[-1][0]
'a'
```

- a[-1]로 리스트 ['a', 'b', 'c']에 접근하고, [0]으로 요소 'a"에 접근

02-3 리스트 자료형

▪ 리스트 슬라이싱

- 문자열과 같이 슬라이싱 적용 가능

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2]
```

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:2]  ◀── 처음부터 a[1]까지
>>> c = a[2:]  ◀── a[2]부터 마지막까지
>>> b
[1, 2]
>>> c
[3, 4, 5]
```

02-3 리스트 자료형

▪ 리스트 연산하기

▪ 더하기(+)

- + 기호는 2개의 리스트를 합치는 기능
- 문자열에서 "abc" + "def" = "abcdef"가 되는 것과 같은 의미

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

▪ 반복하기(*)

- * 기호는 리스트의 반복을 의미
- 문자열에서 "abc" * 3 = "abcabcabc"가 되는 것과 같은 의미

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

02-3 리스트 자료형

▪ 리스트 연산하기

▪ 리스트 길이 구하기

- len( ) 함수 사용
- 문자열, 리스트 외에 앞으로 배울 튜플과 딕셔너리에서도 사용 가능한 내장 함수

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

02-3 리스트 자료형

▪ 리스트의 수정과 삭제

▪ 리스트에서 값 수정하기

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

▪ 리스트 요소 삭제하기

- [ ] 사용해 리스트 요소 삭제하기

```
>>> a = [1, 6, 7, 4]
>>> a[1:3] = []
>>> a
[1, 4]
```

- del 키워드 사용

del 객체

```
>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
```

```
>>> a = [1, 2, 3, 4, 5]
>>> del a[2:]
>>> a
[1, 2]
```

※ 슬라이싱 기법 활용 가능

02-3 리스트 자료형

▪ 리스트 관련 함수

▪ append( )

- 리스트의 맨 마지막에 요소 추가

```
>>> a = [1, 2, 3]
>>> a.append(4) ◀── 리스트의 맨 마지막에 4를 추가
>>> a
[1, 2, 3, 4]
```

- 어떤 자료형도 추가 가능

```
>>> a.append([5,6]) ◀── 리스트의 맨 마지막에 [5,6]을 추가
>>> a
[1, 2, 3, 4, [5, 6]]
```

▪ sort( )

- 리스트의 요소를 순서대로 정렬

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

- 문자의 경우 알파벳 순서로 정렬 가능

02-3 리스트 자료형

▪ 리스트 관련 함수

▪ reverse( )

- 리스트를 역순으로 뒤집어 줌
- 요소를 역순으로 정렬하는 것이 아닌, 현재의 리스트 그대로 뒤집음

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

▪ index( )

- 요소를 검색하여 위치 값 반환

```
>>> a = [1,2,3]
>>> a.index(3) ◀── 3은 리스트 a의 세 번째(a[2]) 요소
2
```

```
>>> lst = ['red', 'blue', 'black', 'blue']
>>> lst.index('blue')
1
>>> lst.index('blue', 2)
3
>>> lst.index('blue', 1, 3)
```

- 값이 존재하지 않으면, 값 오류 발생

```
>>> a.index(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 0 is not in list
```

오류 메시지

02-3 리스트 자료형

▪ 리스트 관련 함수

- **insert( )**
  - 리스트에 요소 삽입
  - insert(a, b)
    - a번째 위치에 b를 삽입하는 함수

```
>>> a = [1, 2, 3]
>>> a.insert(0, 4) ◀── a[0] 위치에 4 삽입
[4, 1, 2, 3]
```

- **remove( )**
  - remove(x)
    - 리스트에서 첫 번째로 나오는 x를 삭제

```
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
[1, 2, 1, 2, 3]
```

  - 값이 여러 개인 경우 첫 번째 것만 삭제

```
>>> a.remove(3)
[1, 2, 1, 2]
```

02-3 리스트 자료형

▪ 리스트 관련 함수

- **pop( )**
  - 리스트의 맨 마지막 요소를 돌려주고 해당 요소 삭제
  - pop(x)
    - 리스트의 x번째 요소를 돌려주고 해당 요소 삭제

```
>>> a = [1, 2, 3]
>>> a.pop()
3
>>> a
[1, 2]
```

```
>>> a = [1, 2, 3]
>>> a.pop(1)
2
>>> a
[1, 3]
```

- **count( )**
  - 리스트에 포함된 요소의 개수 반환
  - count(x)
    - 리스트 안에 x가 몇 개 있는지 조사하여 그 개수를 돌려주는 함수

```
>>> a = [1, 2, 3, 1]
>>> a.count(1)
2
```

02-3 리스트 자료형

▪ 리스트 관련 함수

- **extend( )**
  - 리스트에 리스트를 더하는 함수
  - extend(x)
    - x에는 리스트만 올 수 있음

a.extend([4, 5])

=

a += [4, 5]

```
>>> a = [1, 2, 3]
>>> a.extend([4, 5])
>>> a
[1, 2, 3, 4, 5]
>>> b = [6, 7]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

02-4 튜플 자료형

▪ 튜플(Tuple)이란?

▪ 리스트와 유사한 자료형

리스트	튜플
[ ]로 둘러쌈	( )로 둘러쌈
생성 / 삭제 / 수정 가능	값 변경 불가능

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

- 튜플은 1개의 요소만을 가질 때는 요소 뒤에 콤마(,)를 만드시 붙여야 함 (예) t2 = (1,)
- 괄호( )를 생략해도 무방함 (예) t4 = 1, 2, 3
- 프로그램이 실행되는 동안 값을 유지해야 한다면 튜플을, 수시로 값을 변경해야 하면 리스트 사용

02-4 튜플 자료형

▪ 튜플의 요솟값을 지울 수 있을까?

- 튜플의 요솟값은 한 번 정하면 지우거나 변경할 수 없음!

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]  ◀── 튜플 t1의 첫 번째 요소를 지우려고 시도
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

형 오류(Type Error) 발생

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'  ◀── 튜플 t1의 첫 번째 요솟값을 변경하려고 시도
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

형 오류 발생

02-4 튜플 자료형

튜플 다루기

인덱싱하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

슬라이싱하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:] ◀── t1[1]부터 끝까지
(2, 'a', 'b')
```

튜플 더하기와 곱하기

```
>>> t2 = (3, 4)
>>> t1 + t2
(1, 2, 'a', 'b', 3, 4)
```

```
>>> t2 * 3
(3, 4, 3, 4, 3, 4)
```

튜플 길이 구하기

- len( ) 함수

```
>>> t1 = (1, 2, 'a', 'b')
>>> len(t1)
4
```

02-5 딕셔너리 자료형

딕셔너리(Dictionary)란?

- 대응 관계를 나타내는 자료형
- 연관 배열(Associative aray) 또는 해시(Hash)
- Key와 Value를 한 쌍으로 갖는 자료형
- 순차적으로 해당 요솟값을 구하지 않고, Key를 통해 Value를 바로 얻는 특징
- key는 중복 및 변경 불가

02-5 딕셔너리 자료형

▪ 딕셔너리의 모습

```
{Key1:Value1, Key2:Value2, Key3:Value3, ···}
```

- Key와 Value의 쌍 여러 개 (Key : Value)
- Key : 변하지 않는 값(숫자, 문자열, 튜플)을 사용
- Value : 변하는 값과 변하지 않는 값 모두 사용

- 순서이 개념이 존재하지 않아 인덱싱 및 슬라이싱 불가능

- { }로 둘러싸임
- 각 요소는 쉼표(,)로 구분됨

```
>>> dic = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
```

```
>>> a = {1: 'hi'}
```

```
>>> a = {'a': [1,2,3]}
```

02-5 딕셔너리 자료형

▪ 딕셔너리 쌍 추가, 삭제하기

▪ 딕셔너리 쌍 추가

```
>>> a = {1: 'a'}
>>> a[2] = 'b' ◀── {2:'b'} 쌍 추가
>>> a
{1: 'a', 2: 'b'}
```

▪ 딕셔너리 요소 삭제

```
>>> del a[1] ◀── key가 1인 key:value 쌍 삭제
>>> a
{'name': 'pey', 3: [1, 2, 3], 2: 'b'}
```

▪ 딕셔너리에서 Key 사용해 Value 얻기

```
>>> grade = {'pey': 10, 'julliet': 99}
>>> grade['pey'] ◀── Key가 'pey'인 딕셔너리의 Value를 반환
10
>>> grade['julliet'] ◀── Key가 'julliet'인 딕셔너리의 Value를 반환
99
```

- 리스트나 튜플, 문자열은 요솟값 접근 시 **인덱싱**이나 **슬라이싱** 기법을 사용
- 딕셔너리는 **Key**를 사용해 **Value** 접근

02-5 딕셔너리 자료형

- 딕셔너리 만들 때 주의할 사항
  - key 중복 금지

```
>>> a = {1:'a', 1:'b'}
>>> a
{1: 'b'}
```

  - key에 리스트 사용 금지

```
>>> a = {[1,2] : 'hi'}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

02-5 딕셔너리 자료형

▪ 딕셔너리 관련 함수

▪ keys( )

- Key만을 모아서 dict_keys 객체 반환

```
>>> a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}
>>> a.keys()
dict_keys(['name', 'phone', 'birth'])
```

- 리스트처럼 사용할 수 있지만,
  리스트 관련 함수(append, insert, pop 등)는
  사용 불가능

```
>>> for k in a.keys():
...     print(k)
...
name
phone
birth
```

- dict_keys 객체를 리스트로 변환하는 방법

```
>>> list(a.keys())
['name', 'phone', 'birth']
```

02-5 딕셔너리 자료형

▪ 딕셔너리 관련 함수

▪ values( )

- Value만을 모아서 dict_values 객체 반환

```
>>> a.values()
dict_values(['pey', '0119993323', '1118'])
```

▪ items( )

- Key와 Value의 쌍을 튜플로 묶은 값을 모아서 dic_items 객체 반환

```
>>> a.items()
dict_items([('name', 'pey'), ('phone', '0119993323'), ('birth', '1118')])
```

02-5 딕셔너리 자료형

▪ 딕셔너리 관련 함수

▪ clear( )

- 딕녀서리 내의 모든 요소 삭제
- 빈 딕셔너리는 { }로 표현

```
>>> a.clear()
>>> a
{}
```

▪ in

- Key가 딕셔너리 안에 있는지 조사
- Key가 딕셔너리 안에 존재하면 True, 존재하지 않으면 False 반환

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> 'name' in a
True
>>> 'email' in a
False
```

02-5 딕셔너리 자료형

▪ 딕셔너리 관련 함수

▪ get( )

- Key에 대응되는 Value 반환

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> a.get('name')
'pey'
>>> a.get('phone')
'0119993323'
```

- 존재하지 않는 키 사용 시 None 반환
  - 오류를 발생시키는 list와 차이가 있음

```
>>> a = {'name':'pey', 'phone':'0119993323', 'birth': '1118'}
>>> print(a.get('nokey')) ◀── None을 리턴함
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

- Key 값이 없을 경우
  디폴트 값을 대신 반환하도록 지정 가능

```
>>> a.get('foo', 'bar')
'bar'
```

02-6 집합 자료형

집합(Set)이란?

- 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형
- 파이썬 2.3부터 지원

- set 키워드를 사용하여 생성
  - set( )에 리스트를 입력하여 생성

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
```

set( )에 문자열을 입력하여 생성

```
>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}
```

- 비어 있는 집합 자료형은 s = set()로 만들 수 있음

02-6 집합 자료형

▪ 집합의 특징

1) 중복을 허용하지 않음
2) 순서가 없음(Unordered)

```
>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}
```

- 리스트나 튜플은 순서가 있기 때문에 인덱싱을 통해 자료형의 값을 얻지만 set 자료형은 순서가 없기 때문에 인덱싱 사용 불가 (딕셔너리와 유사)

- 인덱싱 사용을 원할 경우 리스트나 튜플로 변환 필요
  - list( ), tuple( ) 함수 사용

```
>>> t1 = tuple(s1) ◀── 튜플로 변환
>>> t1
(1, 2, 3)
>>> t1[0]
1
```

```
>>> s1 = set([1,2,3])
>>> l1 = list(s1) ◀── 리스트로 변환
>>> l1
[1, 2, 3]
>>> l1[0]
1
```

02-6 집합 자료형

▪ 교집합, 합집합, 차집합 구하기

- set 자료형을 유용하게 사용할 수 있음

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

- 교집합
  - '&' 기호나 intersection( ) 함수 사용

```
>>> s1 & s2
{4, 5, 6}
```

```
>>> s1.intersection(s2)
{4, 5, 6}
```

- 합집합
  - ' | ' 기호나 union( ) 함수 사용

```
>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- 차집합
  - ' – ' 기호나 difference( ) 함수 사용

```
>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}
```

```
>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

02-6 집합 자료형

▪ 집합 관련 함수

▪ add( )

- 이미 만들어진 set 자료형에 값 추가

```
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}
```

▪ update( )

- 여러 개의 값을 한꺼번에 추가

```
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}
```

▪ remove( )

- 특정 값을 제거

```
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

02-7 불 자료형

불(Bool)이란?

- 참(True)과 거짓(False)을 나타내는 자료형

```
>>> a = True
>>> b = False
```

- type( ) 함수를 사용하여 자료형 확인

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

- 조건문의 반환 값으로도 사용됨

```
>>> 1 == 1
True
```

```
>>> 2 > 1
True
```

```
>>> 2 < 1
False
```

02-7 불 자료형

▪ 자료형의 참과 거짓

자료형	값	참 or 거짓
문자열	"python"	참
	""	거짓
리스트	[1,2,3]	참
	[]	거짓
튜플	()	거짓
딕셔너리	{}	거짓
숫자형	0이 아닌 숫자	참
	0	거짓
	None	거짓

02-7 불 자료형

▪ 불 연산

- bool( ) 함수
  - 자료형의 참/거짓을 식별하는 내장 함수
  - 예제 1: 문자열의 참/거짓
    - 'python' 문자열은 빈 문자열이 아니므로 bool 연산의 결과로 True 반환

```
>>> bool('python')
True
```

    - ' ' 문자열은 빈 문자열이므로 bool 연산의 결과로 False 반환

```
>>> bool('')
False
```

  - 예제 2: 리스트, 숫자의 참/거짓

```
>>> bool([1,2,3])
True
>>> bool([])
False
>>> bool(0)
False
>>> bool(3)
True
```

02-7 불 자료형

- **자료형의 참과 거짓은 어떻게 사용되나?**

  - 논리 연산자를 통한 논리 연산이나, 수치를 비교할 때 사용하는 비교연산자의 결과로 사용

```
>>> 3 > 5
False
>>> 4 < 6
True
>>> 'a'=='b'
False
>>> 3.14!=3.14
False
```

02-8 자료형의 값을 저장하는 공간, 변수

▪ 변수란?

- 파이썬에서 사용하는 변수는
  객체를 가리키는 것이라고 할 수 있음
  - 객체 = 자료형과 같은 것을 의미하는 말

```
>>> a = 1
>>> b = "python"
>>> c = [1,2,3]
```

- a, b, c를 변수라고 함

- 변수 선언 방법
  - =(assignment) 기호 사용

변수 이름 = 변수에 저장할 값

```
>>> 4pple = 10
>>> admin = 50
>>> Apple = 60
>>> apple = 70
>>> print(admin, Apple, apple)
>>> _4pple = 10
```

02-8 자료형의 값을 저장하는 공간, 변수

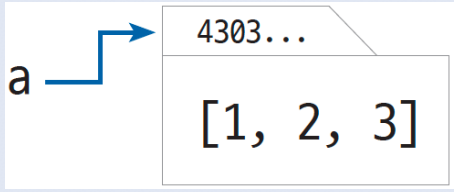
▪ 변수란?

▪ 변수의 예시

```
>>> a = [1, 2, 3]
```

- [1, 2, 3] 값을 가지는 리스트 자료형(객체)이 자동으로 메모리에 생성됨
- 변수 a는 [1, 2, 3] 리스트가 저장된 메모리의 주소를 가리킴
  - id( ) 함수를 사용하여 메모리 주소 확인

```
>>> a = [1, 2, 3]
>>> id(a)
4303029896
```




```
>>> a = [1,2,3]
>>> b = a
>>> id(a)
4303029896
>>> id(b)
4303029896
>>> a is b # a와 b가 가리키는 객체는 동일한가?
True
```

02-8 자료형의 값을 저장하는 공간, 변수

```
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 4, 3]
```

- b 변수를 생성할 때 a 변수의 값을 가져오면서 a와는 다른 주소를 가리키도록 만들수는 없을까?

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 2, 3]
```

```
>>> from copy import copy
>>> b = copy(a)
```

```
>>> b is a
False
```

02-8 자료형의 값을 저장하는 공간, 변수

▪ 변수를 만드는 여러 가지 방법

▪ 튜플

```
>>> a, b = ('python', 'life')
```

- 괄호 생략 가능

```
>>> (a, b) = 'python', 'life'
```

▪ 리스트

```
>>> [a,b] = ['python', 'life']
```

▪ 변수 값 바꾸기

```
>>> a = 3
>>> b = 5
>>> a, b = b, a ◀── a와 b의 값을 바꿈
>>> a
5
>>> b
3
```

02-8 자료형의 값을 저장하는 공간, 변수

- **변수명 작성시 유의사항**
  - 대소문자 구별
  - 변수의 형(type)을 선언할 필요도 없음
  - 변수명은 숫자, 알파벳, 밑줄 기호(_), 한글 등을 사용할 수 있음
  - 변수명은 숫자로 시작하면 안되고, 공백을 포함하면 안됨
  - 키워드(예약어) 사용 금지

```
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
```

02-9 입/출력 함수

▪ 입력함수(input()) 기본 활용법

```
print("Enter your name:")
x = input()
print("Hello, " + x)
```

▪ 2개 이상의 값 입력

```
변수1, 변수2 = input().split()
변수1, 변수2 = input().split('기준문자열')
변수1, 변수2 = input('문자열').split()
변수1, 변수2 = input('문자열').split('기준문자열')
```

02-9 입/출력 함수

- input() 함수 입력 값은 문자열임
- 사칙 연산 계산을 위해서는 형변환(casting) 필요

```
x = int(input())
```

- map함수 사용하여 형 변환하기

```
변수1, 변수2 = map(자료형, input().split())
변수1, 변수2 = map(자료형, input().split('기준문자열'))
변수1, 변수2 = map(자료형, input('문자열').split())
변수1, 변수2 = map(자료형, input('문자열').split('기준문자열'))
```

02-9 입/출력 함수

▪ print() 함수 기본 문법

```
print(값1, 값2, 값3)
print(변수1, 변수2, 변수3)
```

```
>>> print(1, 2, 3)
1 2 3
```

▪ 출력값 사이에 문자 넣기

```
print(값1, 값2, sep='문자 또는 문자열')
print(변수1, 변수2, sep='문자 또는 문자열')
```

```
>>> print(1, 2, 3, sep=', ')
>>> print(4, 5, 6, sep=',')
>>> print('Hello', 'Python', sep='')
>>> print(1920, 1080, sep='x')
```

02-9 입/출력 함수

```
>>> print(1, 2, 3, sep='\n')
1
2
3
```

```
>>> print(1, end=' ')
>>> print(2, end=' ')
>>> print(3)
```

▪ format 함수 사용 출력하기

```
>>> print("I eat %d apples" %3)
>>> print("I eat {} apples" .format(3))
```

```
>>> print("I eats %d apples and %d bananas" %(3, 5))
>>> print("I eats {} apples and {} bananas" .format(3, 5))
```

```
>>> print(" π is %.1f " %(3.14))
>>> print(" π is {} " .format(3.14))
```

02-9 입/출력 함수

- # 한줄 주석
- """ ~""" 여러줄 주석

```
#This is a comment.
print("Hello, World!")
```

```
print("Hello, World!") #This is a comment.
```

```
"""This is a
 multiline
 docstring. """
 print("Hello, World!")
```

감사합니다

“Life is too short,

You need Python!”

인생은 너무 짧으니,

파이썬이 필요해!