2장. 프로그래밍의 기초, 자료형

Contents

I. 숫자형

숫자형(Numeric)이란 숫자 형태로 이루어진 자료형

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

I. 숫자형

❖ 숫자형은 어떻게 만들고 사용할까?

- 정수형(Integer)이란 정수를 뜻하는 자료형을 말함

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

I. 숫자형

- 실수형(Floating-point)은 소수점이 포함된 숫자를 말함

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

✓ 실수형의 소수점 표현 방식

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

✓ 컴퓨터식 지수 표현 방식

✓ 4.24E10은 $4.24*10^{10}$

✓ e와 E 둘 중 어느 것을 사용해도 무방함

I. 숫자형

- 8진수와 16진수

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

✓ 8진수(Octal)를 만들기 위해서는 숫자가 0o 또는 0O로 시작함

```
>>> a = 0x5ff
>>> b = 0xABC
```

✓ 16진수(Hexadecimal)를 만들기 위해서는 0x로 시작함

I. 숫자형

❖ 숫자형을 활용하기 위한 연산자

- 사칙연산(+, -, *, /)

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a - b
-1
>>> a * b
12
>>> a / b
0.75
```

- 대입연산(+=, -=, *=, /=)

```
>>> a += 1
4
```

- x의 y제곱을 나타내는 연산자

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

- 나눗셈 후 나머지를 반환하는 % 연산자

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

- 나눗셈 후 몫을 반환하는 // 연산자

```
>>> 7 // 3
2
>>> 7 // 4
1
>>> 7 / 4
1.75
```

II. 문자열 자료형

문자열(String)이란 문자, 단어 등으로 구성된 문자들의 집합을 의미

❖ 문자열 자료형 만드는 4가지 방법

1. 큰따옴표(")로 양쪽 둘러싸기

```
"Hello World"
```

2. 작은따옴표(')로 양쪽 둘러싸기

```
'Python is fun'
```

3. 큰따옴표 3개를 연속(""")으로 써서 양쪽 둘러싸기

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

4. 작은따옴표 3개를 연속(''')으로 써서 양쪽 둘러싸기

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

- **문자열에 따옴표 포함시키기**

1. 문자열에 작은따옴표(') 포함시키기

   ✓ 문자열을 큰따옴표(")로 둘러싸야 함

   ✓ 큰따옴표 안에 들어 있는 작은 따옴표는 문자열을 나타내기 위한 기호로 인식하지 않음

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

   ✓ 변수에 저장된 문자열이 그대로 출력되는 것을 확인 할 수 있음

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

- ✓ 큰따옴표(")가 아닌 작은따옴표(')로 문자열을 둘러싼 후 다시 실행함
- ✓ 'Python'이 문자열로 인식되어 구문 오류(Syntax Error)가 발생함

```
>>> food = 'Python's favorite food is perl'
File "<stdin>", line 1
        food = 'Python's favorite food is perl'
                        ^
SyntaxError L invalid syntax
```

2. 문자열에 큰따옴표(") 포함시키기

✓ 문자열을 작은따옴표(')로 둘러싸야 함

✓ 작은따옴표(') 안에 사용된 큰따옴표는(")는 문자열을 만드는 기호로 인식되지 않음

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

3. 백슬래시(₩)를 이용해서 작은따옴표(')와 큰따옴표(")를 문자열에 포함시키기

✓ 백슬래시(₩)를 작은따옴표(')나 큰따옴표(") 앞에 삽입하면 백슬래시(₩) 뒤의 작은따옴표(')나 큰따옴표(")는 문자열을 둘러싸는 기호의 의미가 아니라 ('), (") 그 자체를 의미함

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

- **여러 줄인 문자열을 변수에 대입하고 싶을 때**

1. 줄을 바꾸는 이스케이프 코드 '₩n' 삽입하기

```
>>> multiline = "Life is too short₩nYou need python"
>>> print(multiline)
Life is too short
You need python
```

✓ 줄바꿈 문자인 '₩n'을 삽입하는 방법이 있지만, 읽기에 불편하고 줄이 길어지는 단점이 있음

[이스케이프 코드]

프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 '문자 조합'이며, 출력물을 보기 좋게 정렬해 줌

코드	설명
₩n	문자열 안에서 줄을 바꿀 때 사용
₩t	문자열 사이에 탭 간격을 줄 때 사용
₩₩	문자 ₩를 그대로 표현할 때 사용
₩'	작은따옴표(')를 그대로 표현할 때 사용
₩"	큰따옴표(")를 그대로 표현할 때 사용

2. 연속된 작은따옴표 3개(‘’’) 또는 큰따옴표 3개(”””) 이용

✓ 줄바꿈 문자인 ‘₩n’의 단점을 극복하기 위해 사용

✓ 작은따옴표 3개를 사용한 경우

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

✓ print(multiline)을 입력해서 출력을 확인함

```
>>> print(multiline)
Life is too short
You need python
```

❖ 문자열 연산하기

1. 문자열 더해서 연결하기

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

2. 문자열 곱하기

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

- ✓ *는 문자열의 반복을 뜻하는 의미로 사용
- ✓ a * 2는 두 번 반복하라는 뜻

3. 문자열 곱하기 응용

```
#multistring.py

print("=" * 50)
print("My Program")
print("=" * 50)
```

결과값은 다음과 같음

```
C:₩doit>python multistring.py
==================================================
My Program
==================================================
```

4. 문자열 길이 구하기

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

✓ **len( )** : 문자열의 길이를 구해주는 기본 내장 함수

❖ 문자열 인덱싱과 슬라이싱

인덱싱(Indexing)이란 무엇인가를 가리킨다는 의미이고, 슬라이싱(Slicing)은 무엇인가를 잘라낸다는 의미임

▪ 문자열 인덱싱이란?

```
>>> a = "Life is too short, You need Python"
```

✓ 변수 a에 저장한 문자열의 각 문자마다 번호를 매겨보면 다음과 같음

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

✓ 문자열에서 L은 첫 번째 자리를 뜻하는 숫자인 0, 바로 다음인 i는 1 이런 식으로 계속 번호를 붙인 것임

```
>>> a = "Life is too short, You need Python"
>>> a[3]
'e'
```

- ✓ a[3]이 뜻하는 것은 a라는 문자열의 네 번째 문자인 e를 말함
- ✓ 파이썬은 0부터 숫자를 셈

▪ 문자열 인덱싱 활용하기

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

✓ a[-1]은 문자열을 뒤에서부터 읽기 위해서 마이너스(-)를 붙임

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

✓ a[-0]은 a[0]과 똑같은 값을 보여줌

```
>>> a[-2]
'o'
>>> a[-5]
'y'
```

✓ a[-2]는 뒤에서 두 번째 문자를 가리키는 것이고, a[-5]는 뒤에서 다섯 번째 문자를 가리키는 것

- **문자열 슬라이싱이란?**

```
>>> a = "Life is too short, You need Python"
>>> b = a[0] + a[1] + a[2] + a[3]
>>> b
'Life'
```

  - ✓ 단순히 한 문자만을 뽑아내는 것이 아니라 'Life' 또는 'You' 같은 단어를 뽑아 내는 방법
  - ✓ 위와 같이 단순하게 접근할 수 있지만, 슬라이싱이라는 더 좋은 방법을 제공함

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

- ✓ 슬라이싱 기법으로 간단하게 처리할 수 있음
- ✓ **a[시작 번호:끝 번호]**를 지정하면, a문자열에서 자리번호 0부터 4까지의 문자를 뽑아낸다는 뜻
- ✓ 끝 번호에 해당하는 것은 포함되지 않는 것에 주의
- ✓ 수식으로 나타내면 아래와 같음

```
0 <= a < 4
```

- ✓ a는 a[0], a[1], a[2], a[3]임

- **문자열을 슬라이싱하는 방법**

```
>>> a[0:5]
'Life '
```

✓ 위 예는 a[0] + a[1] + a[2] + a[3] + a[4]와 동일

✓ a[4]는 공백 문자이기 때문에 'Life' 가 아닌 'Life '가 출력됨

```
>>> a[0:2]
'Li'
>>> a[5:7]
'is'
>>> a[12:17]
'short'
```

✓ 슬라이싱할 때 항상 시작 번호가 '0'일 필요는 없음

```
>>> a[19:]
'You need Python'
```

✓ a[시작 번호:끝 번호]에서 끝 번호 부분을 생략하면 시작 번호부터 그 문자열의 끝까지 뽑아냄

```
>>> a[:17]
'Life is too short'
```

✓ a[시작 번호:끝 번호]에서 시작 번호 부분을 생략하면 처음부터 끝 번호까지 뽑아냄

```
>>> a[:]
'Life is too short, You need Python'
```

✓ a[시작 번호:끝 번호]에서 시작 번호와 끝 번호를 생략하면 처음부터 끝까지 뽑아냄

```
>>> a[19:-7]
'You need'
```

- ✓ 슬라이싱에도 인덱싱과 마찬가지로 마이너스(-) 기호를 사용할 수 있음
- ✓ 위 소스 코드에서 a[19:-7]이 뜻하는 것은 a[19]에서부터 a[-8]까지를 의미함
- ✓ a[-7]은 포함하지 않음

- **슬라이싱으로 문자열 나누기**

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

- ✓ a라는 문자열을 두 부분으로 나누는 기법임
- ✓ 동일한 8을 기준으로 a[:8], a[8:]처럼 사용, a[:8]은 a[8]을 포함하지 않고 a[8:]은 포함하기 때문에 두 부분으로 나눌 수 있음

```
>>> a = "20010331Rainy"
>>> year = a[:4]
>>> day = a[4:8]
>>> weather = a[8:]
>>> year
'2001'
>>> day
'0331'
>>> weather
'Rainy'
```

- ✓ 위의 문자열 "20010331Rainy"를 연도인 2001, 월과 일을 나타내는 0331, 날씨를 나타내는 Rainy의 세 부분으로 나눔

- **문자열 바꾸기**
  - ✓ 'Pithon' 이라는 문자열을 'Python'으로 바꾸려면?

```
>>> a = 'Pithon'
>>> a[1]
'i'
>>> a[1] = 'y'
```

  - ✓ 문자열의 요소 값은 바꿀 수 있는 값이 아니기 때문에 에러가 발생함

```
>>> a = 'Pithon'
>>> a[:1]
'P'
>>> a[2:]
'thon'
>>> a[:1] + 'y' + a[2:]
'Python'
```

  - ✓ 'Pithon'이라는 문자열을 'P' 부분과 'thon' 부분으로 나눌 수 있기 때문에 그 사이에 'y'라는 문자를 추가하여 'Python'이라는 새로운 문자열을 만들 수 있음

❖ 문자열 포매팅

"현재 온도는 18도입니다."

시간이 지나서 20도가 되면 아래와 같은 문장으로 출력한다.

"현재 온도는 20도입니다."

- 위의 두 문자열은 모두 같은데 20이라는 숫자와 18이라는 숫자만 다름
- 이렇게 문자열 내의 특정한 값을 바꿔야 할 경우가 있을 때 가능하게 해주는 것이 문자열 포매팅 기법임
- 문자열 포매팅(Formatting)이란 문자열 내의 어떤 값을 삽입하는 방법임

- **문자열 포매팅 따라하기**

1. 숫자 바로 대입

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

✓ 문자열 안에서 숫자를 넣고 싶은 자리에 %d라는 문자를 넣어주고, 삽입할 숫자인 3은 가장 뒤에 있는 %문자 다음에 써넣음

✓ %d는 문자열 포맷 코드라 부름

2. 문자열 바로 대입

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

✓ 문자열 내에 또 다른 문자열을 삽입하기 위해 문자열 포맷 코드 %s를 씀

3. 숫자 값을 나타내는 변수로 대입

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

✓ 숫자 값을 나타내는 변수를 대입할 수 있음

4. 2개 이상의 값 넣기

```
>>> 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.'
```

✓ 마지막 % 다음 괄호 안에 콤마(,)로 구분하여 각각의 값을 넣어 주면 됨

- **문자열 포맷 코드**

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

✓ %s 포맷 코드는 어떤 형태의 값이든 변환해서 넣을 수 있음

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

✓ 3을 문자열 안에 삽입하려면 %d를 사용하고, 3.234를 삽입하려면 %f를 사용해야 함

✓ 하지만 %s를 사용하면 이런 것을 생각하지 않아도 됨. %s는 자동으로 % 뒤에 있는 값을 문자열로 바꾸기 때문임

✓ 포맷팅 연산자 %d와 %를 같이 쓸 때는 %%를 씀

```
>>> "Error is %d%." % 98
Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
ValueError: incomplete format
```

✓ 문자열 포맷팅 코드인 %d와 %가 문자열 내에 존재하는 경우, %를 나타내려면 반드시 %%로 써야함

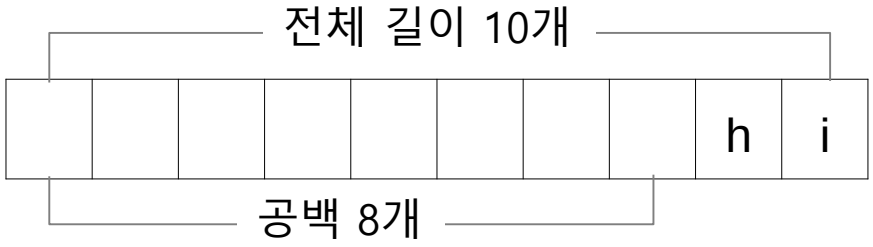
```
>>> "Error is %d%%." % 98
'Error is 98%.'
```

▪ 포맷 코드와 숫자 함께 사용하기

1. 정렬과 공백

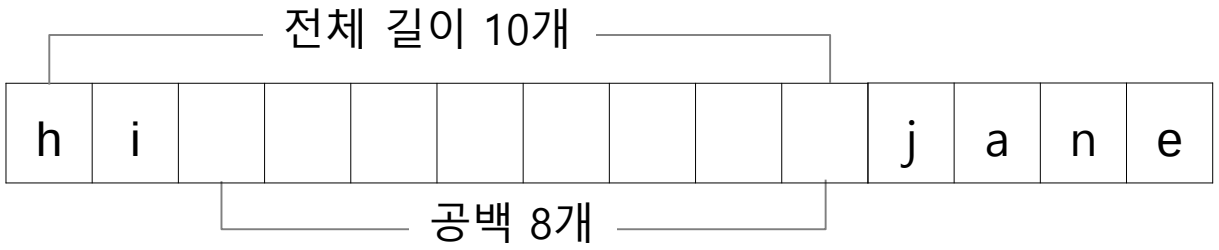
```
>>> "%10s" % "hi"
'        hi'
```

- ✓ hi가 오른쪽 정렬됨
- ✓ %10s는 전체 길이가 10인 문자열 공간에서 대입되는 값을 오른쪽으로 정렬하고, 그 앞의 나머지는 공백으로 남겨두라는 의미임



```
>>> "%-10sjane" % "hi"
'hi         jane'
```

- ✓ 반대쪽인 왼쪽 정렬은 "%-10s"가 될 것임
- ✓ hi를 왼쪽으로 정렬하고 나머지는 공백으로 채움



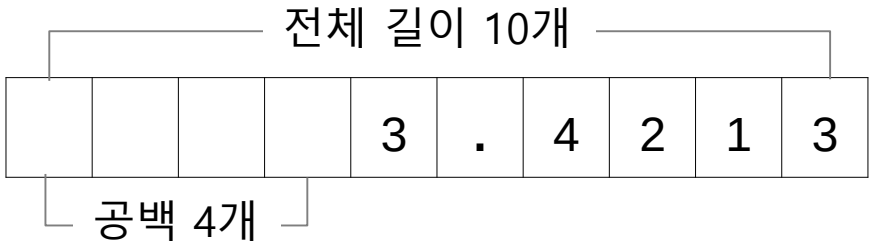
2. 소수점 표현하기

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

- ✓ '.'의 의미는 소수점 포인트를 의미하고, 그 뒤의 숫자 4는 소수점 뒤에 나올 숫자의 개수를 말함

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

- ✓ 소수점 포인트 앞의 숫자 10은 전체 길이의 숫자를 말함
- ✓ 소수점 네 번째 자리까지만 표시하고, 전체 길이가 10개인 문자열 공간에서 오른쪽으로 정렬함



- **format 함수를 사용한 포매팅**

  좀 더 발전된 스타일로 문자열 포맷을 지정할 수 있음

  ✓ 숫자 바로 대입하기

```
>>> "I eat {0} apples".format(3)"
'I eat 3 apples'
```

  - 문자열 중 {0} 부분이 숫자 3으로 바뀜

  ✓ 문자열 바로 대입하기

```
>>> "I eat {0} apples".format("five")
'I eat five apples'
```

  - 문자열 중 {0} 부분이 five라는 문자열로 바뀜

✓ 숫자 값을 가진 변수로 대입하기

```
>>> number = 3
>>> "I eat {0} apples".format(number)
'I eat 3 apples'
```

- 문자열의 {0} 항목이 number 변수 값인 3으로 바뀜

✓ 2개 이상의 값 넣기

```
>>> number = 10
>>> day = "three"
>>> "I ate {0} apples . so I was sick for {1} days.".format(number, day)
'I ate 10 apples. so I was sick for three days.'
```

- 문자열의 {0}, {1}와 같은 인덱스 항목이 format 함수의 입력값으로 순서에 맞게 바뀜

✓ 이름으로 넣기

```
>>> "I ate {number} apples . so I was sick for {day} days.".format(number=10, day=3)
'I ate 10 apples. so I was sick for 3 days.'
```

- {0}, {1}과 같은 인덱스 항목 대신 더 편리한 {name} 형태를 사용함

✓ 인덱스와 이름을 혼용해서 넣기

```
>>> "I ate {0} apples . so I was sick for {day} days.".format(10, day=3)
'I ate 10 apples. so I was sick for 3 days.'
```

- 인덱스 항목과 name=value 형태를 혼용하는 것도 가능함

✓ 왼쪽 정렬

```
>>> "{0:<10}".format("hi")
'hi        '
```

- :< 표현식을 사용하면 치환되는 문자열을 왼쪽으로 정렬하고, 문자열의 총 자릿수를 10으로 맞춤

✓ 오른쪽 정렬

```
>>> "{0:>10}".format("hi")
'        hi'
```

- 오른쪽 정렬은 :>을 사용함.

✓ 가운데 정렬

```
>>> "{0:^10}".format("hi")
'    hi    '
```

- :^ 기호를 사용하면 가운데 정렬함

✓ 공백 채우기

```
>>> "{0:=^10}".format("hi")
'====hi===='
>>> "{0:!<10}".format("hi")
'hi!!!!!!!!'
```

- 공백 문자 대신에 지정한 문자 값으로 채워 넣는 것도 가능함
- 채워 넣을 문자 값은 정렬 문자 <, >, ^ 바로 앞에 넣어야 함

✓ 소수점 표현하기

```
>>> y = 3.42134234
>>> "{0:0.4f}".format(y)
'3.4213'
>>> "{0:10.4f}".format(y)
'    3.4213'
```

- format 함수를 사용해 소수점 4자리까지만 표현하고, 자릿수를 10으로 맞춤

✓ { 또는 } 문자 표현하기

```
>>> "{{ and }}".format()
'{ and }'
```

- { }와 같은 중괄호 문자를 포매팅 문자가 아닌 문자 그대로 사용하고 싶은 경우에는 {{ }} 처럼 2개를 연속해서 사용해야 함

▪ f 문자열 포매팅

파이썬 3.6 버전부터는 f 문자열 포매팅 기능을 사용할 수 있음

✓ 문자열 앞에 f 접두사를 붙여서 사용

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

✓ 표현식 지원

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

✓ 딕셔너리 사용

```
>>> d = {'name':'홍길동', 'age':30}
>>> f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.'
'나의 이름은 홍길동입니다. 나이는 30입니다.'
```

✓ 정렬

```
>>> f'{"hi":<10}'      ---왼쪽 정렬
'hi        '
>>> f'{"hi":>10}'      ---오른쪽 정렬
'        hi'
>>> f'{"hi":^10}'      ---가운데 정렬
'    hi    '
```

✓ 공백 채우기

```
>>> f'{"hi":=^10}'          ---가운데 정렬하고 = 문자로 공백 채우기
'====hi===='
>>> f'{"hi":!<10}'          ---왼쪽 정렬하고 ! 문자로 공백 채우기
'hi!!!!!!!!'
```

✓ 소수점 표현하기

```
>>> y = 3.42134234
>>> f'{y:0.4f}'             ---소수점 4자리까지만 표현
'3.4213'
>>> f'{y:10.4f}'            ---소수점 2자리까지 표현하고, 총 자릿수를 10으로 맞춤
'    3.4213'
```

✓ { 또는 } 문자 표현하기

```
>>> f'{{ and }}'
'{ and }'
```

- { } 문자를 표시하려면, {{ }} 처럼 2개를 연속해서 사용해야 함

❖ 문자열 관련 함수

문자열 자료형은 자체적으로 문자열 내장 함수를 가지고 있으며, 그것들을 사용하려면 **문자열 변수 이름 뒤에 '.'를 붙인 다음에 함수 이름**을 써주면 됨

▪ 문자 개수 세기(count)

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

✓ 문자열 중 문자 b의 개수를 반환함

▪ 위치 알려주기 1(find)

```
>>> a = "Python is the best choice"
>>> a.find('b')
14
>>> a.find('k')
-1
```

✓ 문자열 중 문자 b가 처음으로 나온 위치를 반환하며, 찾는 문자나 문자열이 존재하지 않는다면 -1을 반환함

- **위치 알려주기 2(index)**

```
>>> 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
```

- ✓ 문자열 중 문자 t가 처음으로 나온 위치를 반환하며, 찾는 문자나 문자열이 존재하지 않는다면 오류를 발생시킴
- ✓ find함수와 다른 점은 문자열 안에 존재하지 않는 문자를 찾으면 오류가 발생한다는 점임

- **문자열 삽입(join)**

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

✓ abcd 문자열의 각각의 문자 사이에 ','를 삽입함

- **소문자를 대문자로 바꾸기(upper)**

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

- **대문자를 소문자로 바꾸기(lower)**

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

- **왼쪽 공백 지우기(lstrip)**

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

  - ✓ 문자열 중 가장 왼쪽에 있는 한 칸 이상의 연속된 공백을 모두 지움
  - ✓ lstrip에서 l은 left를 의미하고, rstrip에서 r은 right를 의미함

- **오른쪽 공백 지우기(rstrip)**

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

- **양쪽 공백 지우기(strip)**

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

- **문자열 바꾸기(replace)**

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

✓ **replace(바뀌게 될 문자열, 바꿀 문자열)**를 사용해서 문자열 안의 특정한 값을 다른 값으로 치환해 줌

- **문자열 나누기(split)**

```
>>> a = "Life is too short"
>>> a.split()
['Life', 'is', 'too', 'short']
>>> a = "a:b:c:d"
>>> a.split(':')
['a', 'b', 'c', 'd']
```

- ✓ a.split() 처럼 괄호 안에 아무런 값도 넣어 주지 않으면 공백을 기준으로 문자열을 나누어 줌
- ✓ a.split(':')처럼 괄호 안에 특정한 값이 있을 경우 괄호 안의 값을 구분자로 해서 문자열을 나누어줌
- ✓ 이렇게 나눈 값은 리스트에 하나씩 들어가게 됨

III. 리스트 자료형

리스트(List)란 숫자나 문자 모음으로 이루어진 자료형

❖ 리스트를 어떻게 만들고 사용할까?

- 리스트를 사용하면 1, 3, 5, 7, 9 숫자 모음을 간단하게 표현할 수 있음

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

- 리스트를 만들 때는 대괄호([ ])로 감싸 주고, 각 요소 값들은 쉼표(,)로 구분해 줌

리스트명 = [요소1, 요소2, 요소3, …]

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

- ✓ a처럼 비어있는 리스트([ ])일 수 있음
- ✓ b나 c처럼 숫자나 문자열을 요소 값으로 가질 수 있음
- ✓ d처럼 숫자와 문자열을 함께 요소 값으로 가질 수 있음
- ✓ e처럼 리스트 자체를 요소 값으로 가질 수 있음

❖ 리스트의 인덱싱과 슬라이싱

▪ 리스트의 인덱싱

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

✓ 변수 a에 [1, 2, 3]이라는 값을 설정

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

✓ a[0]은 리스트 a의 첫 번째 요소 값을 의미함

```
>>> a[0] + a[2]
4
```

✓ 리스트의 첫 번째 요소인 a[0]과 세 번째 요소인 a[2]의 값을 더함

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

✓ a[-1]은 리스트 a의 마지막 요소 값을 의미함

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

✓ 리스트 a를 숫자 1, 2, 3과 또 다른 리스트인 ['a', 'b', 'c']를 포함하도록 만듦

```
>>> a[0]
1
>>> a[-1]
['a', 'b', 'c']
>>> a[3]
['a', 'b', 'c']
```

✓ A[-1]은 마지막 요소 값인 ['a', 'b', 'c']를 나타냄

✓ a[3]은 리스트 a의 네 번째 요소로 a[-1]과 동일한 결과값을 보여줌

```
>>> a[-1][0]
'a'
>>> a[-1][1]
'b'
>>> a[-1][2]
'c'
```

✓ 리스트 a에 포함된 ['a', 'b', 'c'] 리스트에서 값을 인덱싱하는 방법임

- **삼중 리스트에서 인덱싱하기**

```
>>> a = [1, 2, ['a', 'b', ['Life', 'is']]]
```

✓ 리스트 a안에 ['a', 'b', ['Life', 'is']] 리스트가 포함되어 있고, 그 리스트 안에 다시 ['Life', 'is'] 리스트가 포함된 삼중 구조의 리스트임

```
>>> a[2][2][0]
'Life'
```

✓ 'Life'라는 문자열을 인덱싱하는 방법임

▪ 리스트의 슬라이싱

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

✓ 문자열의 슬라이싱과 동일함

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

✓ b 변수는 리스트 a의 첫 번째 요소인 a[0]부터 두 번째 요소인 a[1]까지 나타내는 리스트이고, a[2] 값인 3은 포함되지 않음

✓ c 변수는 리스트 a의 세 번째 요소부터 끝까지 나타내는 리스트임

- **중첩된 리스트에서 슬라이싱하기**

```
>>> a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]
>>> a[2:5]
[3, ['a', 'b', 'c'], 4]
>>> a[3][:2]
['a', 'b']
```

✓ a[3]은 ['a', 'b', 'c']를 나타내고, a[3][:2]는 ['a', 'b', 'c']의 첫 번째 요소부터 세 번째 요소 직전까지의 값 ['a', 'b']를 나타내는 리스트가 됨

❖ 리스트 연산하기

리스트도 + 기호를 이용해서 더할 수 있고, * 기호를 이용해서 반복할 수 있음

1. 리스트 더하기(+)

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

✓ 리스트 사이에서 + 기호는 2개의 리스트를 합치는 기능을 함

2. 리스트 반복하기(*)

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

✓ [1, 2, 3] 리스트가 세 번 반복되어 새로운 리스트를 만들어 냄

3. 리스트 길이 구하기

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

- 리스트 연산 오류

```
>>> a = [1, 2, 3]
>>> a[2] + "hi"
Traceback (most recent call last):
  File "stdin", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

✓ 정수와 문자열은 서로 더할 수 없기 때문에 형(type) 오류가 발생함

```
>>> str(a[2]) + "hi"
```

✓ 숫자와 문자열을 더해서 '3hi' 처럼 만들고 싶다면 숫자 3를 문자 '3'으로 바꿈

✓ str 함수는 정수나 실수를 문자열의 형태로 바꾸어 주는 내장 함수임

❖ 리스트의 수정과 삭제

- 리스트에서 값 수정하기

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

✓ a[2]의 요소값 3이 4로 바뀌었음

- del 함수를 사용해 리스트 요소 삭제하기

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

✓ del a[x]는 x번째 요소값을 삭제함. del 객체는 변수를 삭제함

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

- ✓ 슬라이싱 기법을 사용하여 리스트의 요소 여러 개를 삭제함
- ✓ del[x:y]는 x번째부터 y번째 요소 사이의 값을 삭제함

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

- ✓ [ ] 사용해 리스트 요소 삭제함
- ✓ a[1:3]을 [ ]으로 바꿔 주었기 때문에 [‘a’, ‘b’]가 삭제된 [1, ‘c’, 4]가 됨

❖ 리스트 관련 함수

리스트 변수 이름 뒤에 '.'를 붙여서 리스트 관련 함수를 사용할 수 있음

▪ 리스트에 요소 추가(append)

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

- ✓ append(x)는 리스트의 맨 마지막에 x를 추가하는 함수임
- ✓ 리스트 안에는 어떤 자료형도 추가할 수 있음

```
>>> a.append([5, 6])
>>> a
[1, 2, 3, 4, [5, 6]]
```

- ✓ 리스트에 다시 리스트를 추가한 결과임

- **리스트 정렬(sort)**

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

✓ sort 함수는 리스트의 요소를 순서대로 정렬해 줌

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

✓ 문자 역시 알파벳 순서로 정렬할 수 있음

▪ 리스트 뒤집기(reverse)

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

✓ reverse 함수는 리스트를 역순으로 뒤집어 줌

▪ 위치 반환(index)

```
>>> a = [1, 2, 3]
>>> a.index(3)          ---3은 리스트 a의 세 번째(a[2]) 요소
2
>>> a.index(1)          --- 1은 리스트 a의 첫 번째(a[0]) 요소
0
```

✓ index(x) 함수는 리스트에 x라는 값이 있으면 x의 위치 값을 돌려줌

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

✓ 0이라는 값은 a 리스트에 존재하지 않기 때문에 값 오류가 발생함

- **리스트에 요소 삽입(insert)**

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

✓ insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수임

✓ 0번째 자리, 첫 번째 요소(a[0]) 위치에 값 4를 삽입하라는 뜻임

```
>>> a.insert(3, 5)
[4, 1, 2, 5, 3]
```

✓ 3번째 자리, 네 번째 요소(a[3]) 위치에 값 5를 삽입하라는 뜻임

- **리스트에 요소 제거(remove)**

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

  - ✓ remove(x)는 리스트에서 첫 번째 나오는 x를 삭제하는 함수임
  - ✓ a가 3이라는 값을 2개 가지고 있을 경우 첫 번째 3만 제거됨
  - ✓ remove(3)을 한 번 더 실행하면 다시 3이 삭제됨

- **리스트 요소 끄집어 내기(pop)**

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

  - ✓ pop()은 리스트의 맨 마지막 요소를 돌려주고 그 요소는 삭제하는 함수임
  - ✓ a 리스트 [1, 2, 3]에서 3을 끄집어내고 최종적으로 [1, 2]만 남음

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

- ✓ pop(x)는 리스트의 x번째 요소를 돌려주고 그 요소는 삭제함
- ✓ a.pop(1)은 a[1]의 값을 끄집어 내고 값이 삭제된 것이 확인됨

- **리스트에 포함된 요소 x의 개수 세기(count)**

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

- ✓ count(x)는 리스트 내에 x개가 몇 개 있는지 조사하여 그 개수를 돌려주는 함수
- ✓ 1이라는 값이 리스트 a에 2개 들어 있으므로 2를 돌려줌

- **리스트 확장(extend)**

```
>>> 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]
```

- ✓ extend(x)에서 x에는 리스트만 올 수 있으며, a리스트에 x리스트를 더하게 됨
- ✓ a.extent([4, 5])는 a += [4, 5]와 동일함
- ✓ a += [4, 5]는 a = a + [4, 5]와 같음

IV. 튜플 자료형

❖ 튜플(tuple)은 리스트와 거의 비슷하며 다른 점은 다음과 같음

- 리스트는 [ ]으로 둘러싸지만, 튜플은 ( )으로 둘러쌈
- 리스트는 그 값의 생성, 삭제, 수정이 가능하지만, 튜플은 그 값을 바꿀 수 없음

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

- ✓ t2 = (1,)처럼 1개의 요소만을 가질 때는 요소 뒤에 콤마(,)를 반드시 붙여야 함
- ✓ t4 = 1, 2, 3처럼 괄호( )를 생략해도 무방함
- ✓ **프로그램 실행되는 동안 그 값이 변하지 않기를 바란다면 튜플**을 사용하고, **수시로 그 값을 변화시켜야 한다면 리스트**를 사용함

❖ 튜플의 요소값을 지우거나 변경하려고 하면 어떻게 될까?

- 튜플 요소 값을 삭제하려 할 때

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

✓ 튜플은 요소를 지우는 행위가 지원되지 않는다는 메시지를 확인할 수 있음

- 튜플 요소 값을 변경하려 할 때

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

✓ 튜플의 요소 값을 변경하려고 해도 오류가 발생하는 것을 확인할 수 있음

❖ 튜플 다루기

- 인덱싱하기

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

✓ 문자열, 리스트와 같이 t1[0], t1[3]처럼 인덱싱이 가능함

- 슬라이싱하기

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

✓ t1[1]부터 튜플의 마지막 요소까지 슬라이싱 함

- 튜플 더하기

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

- 튜플 곱하기

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

- 튜플의 길이 구하기

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

V. 딕셔너리 자료형

❖ 딕셔너리란?

사람은 누구든지 '이름'='홍길동', '생일'='몇월 몇일' 등으로 구분할 수 있는데, 이러한 대응 관계를 나타낼 수 있는 자료형이 딕셔너리(Dictionary)임

- 딕셔너리는 단어 그대로 해석하면 사전이라는 뜻이며, 연관 배열(Associative array) 또는 해시(Hash)라고 함
- People이라는 단어에 '사람', baseball이라는 단어에 '야구'라는 뜻이 부합되듯이 딕셔너리는 Key와 Value라는 한 쌍을 갖는 자료형임
- Key가 'baseball'이라면, Value는 '야구'가 됨
- 딕셔너리는 리스트나 튜플처럼 순차적으로 해당 요소값을 구하지 않고, Key를 통해 Value를 얻으며 이것이 딕셔너리의 가장 큰 특징임

V. 딕셔너리 자료형

❖ 딕셔너리는 어떻게 만들까?

{Key1:Value1, Key2:Value2, Key3:Value3 …}

- Key와 Value의 쌍 여러 개가 { }로 둘러싸여 있음
- 각각의 요소는 Key : Value 형태로 이루어져 있고 쉼표(,)로 구분되어 있음

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

✓ Key는 각각 'name', 'phone', 'birth'이고, 각 Key에 해당하는 Value는 'pey', '0119993323', '1118'}이 됨

✓ 딕셔너리 dic의 정보

Key	Value
name	pey
phone	0119993323
birth	1118

V. 딕셔너리 자료형

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

✓ Key로 정수값 1, Value로 'hi'라는 문자열을 사용함

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

✓ Value에 리스트도 넣을 수 있음

❖ 딕셔너리 쌍 추가, 삭제하기

1. 딕셔너리 쌍 추가하기

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

✓ {1:'a'} 딕셔너리에 a[2]='b'와 같이 입력하면, 딕셔너리 a에 Key와 Value가 각각 2와 'b'인 2:'b'라는 딕셔너리 쌍이 추가됨

```
>>> a['name'] = 'pey'
>>> a
{'name':'pey', '2:'b', 1:'a'}
```

✓ 딕셔너리 a에 'name':'pey'라는 쌍이 추가됨

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

✓ Key는 3, Value는 [1, 2, 3]을 가지는 한 쌍이 추가됨

2. 딕셔너리 요소 삭제하기

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

- ✓ del 함수를 이용해서 del a[Key]처럼 입력하면 지정한 Key에 해당하는 {Key:Value}쌍이 삭제됨

V. 딕셔너리 자료형

❖ 딕셔너리를 사용하는 방법

각자의 특기를 표현할 때 리스트나 문자열은 표현하기 까다롭지만 딕셔너리는 쉽다.

```
{"김연아":"피겨스케이팅", "류현진":"야구", "박지성":"축구", "귀도":"파이썬"}
```

✓ 사람의 이름과 특기를 한 쌍으로 하는 딕셔너리임

▪ 딕셔너리에서 Key를 사용해 Value 얻기

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

✓ Key의 Value를 얻기 위해서는 '**딕셔너리 변수 이름[Key]**'를 사용함

V. 딕셔너리 자료형

```
>>> a = {1:'a', 2:'b'}
>>> a[1]      --- Key가 1인 요소의 Value를 반환
'a'
>>> a[2]
'b'
```

✓ [ ]안의 숫자 1은 두 번째 요소를 뜻하는 것이 아니라, Key에 해당하는 1을 나타냄

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

✓ 딕셔너리 a는 a[Key]로 입력해서 Key에 해당하는 Value를 얻음

✓ a['a'], a['b']처럼 Key를 사용해 Value를 얻음

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

✓ Key를 사용해서 Value를 얻는 방법을 보여 줌

- **딕셔너리를 만들 때 주의할 사항**

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

✓ 딕셔너리에서 Key는 고유한 값이므로 중복되는 Key 값을 설정해 놓으면 하나를 제외한 나머지 것들이 모두 무시됨. 마지막 Value가 등록됨

✓ Key가 2개 존재할 경우 1:'a'라는 쌍이 무시됨

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

- ✓ Key에 리스트를 쓸 수 없고, 튜플은 Key로 쓸 수 있음
- ✓ 리스트는 그 값이 변할 수 있기 때문에 Key로 쓸 수 없음
- ✓ 딕셔너리의 Key 값으로 딕셔너리를 사용할 수 없음
- ✓ Value에는 변하는 값이든 변하지 않는 값이든 상관없이 아무 값이나 넣을 수 있음

❖ 딕셔너리 관련 함수

▪ Key 리스트 만들기(keys)

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

✓ a.keys( )는 딕셔너리 a의 Key만을 모아서 dict_keys라는 객체를 돌려줌

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

✓ dict_keys 객체는 리스트를 사용하는 것과 차이가 없지만, 리스트 고유의 append, insert, pop, remove, sort 함수는 수행할 수 없음

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

✓ dict_keys 객체를 리스트로 변환함

- **Value 리스트 만들기(values)**

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

✓ values 함수는 딕셔너리 a의 Value만을 모아서 dict_values라는 객체를 돌려줌

- **Key, Value 쌍 얻기(items)**

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

✓ items 함수는 key와 value의 쌍을 튜플로 묶는 값을 dict_items 객체로 돌려줌

- **Key : Value 쌍 모두 지우기(clear)**

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

  - ✓ clear 함수는 딕셔너리 안의 모든 요소를 삭제함
  - ✓ 빈 딕셔너리는 { }로 표현함

- **Key로 Value 얻기(get)**

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

  - ✓ get(x) 함수는 x라는 key에 대응되는 value를 돌려줌
  - ✓ a.get('name')은 a['name']을 사용했을 때와 동일한 결과 값을 돌려 받음

```
>>> print(a.get('nokey'))
None
>>> print(a['nokey'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'nokey'
```

✓ a['nokey']처럼 존재하지 않는 키(nokey)로 값을 가져오려고 할 때 a['nokey']는 Key 오류를 발생시키고, a.get('nokey')는 None을 리턴한다는 차이가 있음

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

✓ 딕셔너리 안에 찾으려는 Key 값이 없을 경우 미리 정해 둔 디폴트 값을 대신 가져오게 하고 싶을 때에는 **get(x, '디폴트 값')**을 사용하면 편리함

- **해당 Key가 딕셔너리 안에 있는지 조사하기(in)**

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

- ✓ 'name' 문자열은 a 딕셔너리의 key 중 하나여서 참(True)을 돌려줌
- ✓ 'email'은 a 딕셔너리 안에 존재하지 않는 Key이므로 거짓(False)을 돌려줌

VI. 집합 자료형

집합에 관련된 것들을 쉽게 처리하기 위해 만들어진 자료형

❖ 집합 자료형은 어떻게 만들까?

- 집합 자료형은 set 키워드를 이용해 만들 수 있음

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
>>> s2 = set("Hello")
>>> s2
{'e', 'l', 'o', 'H'}
```

  - ✓ set( )의 괄호 안에 리스트나 문자열을 입력하여 만들 수 있음

VI. 집합 자료형

❖ 집합 자료형의 특징

- **중복을 허용하지 않음**
- **순서가 없음(Unordered)**
  - 리스트나 튜플은 순서가 있기 때문에 인덱싱을 통해 자료형의 값을 얻을 수 있지만, set 자료형은 순서가 없기 때문에 인덱싱으로 값을 얻을 수 없음
  - set 자료형에 저장된 값을 인덱싱으로 접근하려면 리스트나 튜플로 변환 후 해야 함

```
>>> s1 = set([1, 2, 3])
>>> l1 = list(s1)
>>> l1
[1, 2, 3]
>>> l1[0]
1
>>> t1 = tuple(s1)
>>> t1
(1, 2, 3)
>>> t1[0]
1
```

❖ 교집합, 합집합, 차집합 구하기

- ✓ set 자료형을 유용하게 사용하는 경우는 교집합, 합집합, 차집합을 구할 때임
- ✓ 2개의 set 자료형, s1은 1부터 6까지의 값을 가지게 되었고 s2는 4부터 9까지의 값을 가지게 되었음

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

▪ 교집합(&)

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

- ✓ '&' 기호를 사용하면 교집합을 구할 수 있음

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

- ✓ intersection 함수를 사용해도 동일한 결과를 돌려줌

- **합집합(|)**

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

✓ '|' 기호나 union 함수를 사용하면 합집합을 구할 수 있음

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

- **차집합(-)**

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

✓ 빼기(-) 기호나 difference 함수를 사용하면 차집합을 구할 수 있음

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

❖ 집합 자료형 관련 함수

▪ 값 1개 추가하기(add)

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

✓ 이미 만들어진 set 자료형에 값을 추가할 수 있음

▪ 값 여러 개 추가하기(update)

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

▪ 특정 값 제거하기(remove)

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

VII. 불 자료형

❖ **불(bool) 자료형이란?**

참(True)과 거짓(False)을 나타내는 자료형

- True : 참
- False : 거짓

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

✓ 따옴표로 감싸지 않은 True나 False 예약어를 변수에 지정해서 사용함

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

✓ **type(x)**는 x의 자료형을 확인하는 내장 함수임

```
>>> 1 == 1
True
>>> 2 > 1
True
>>> 2 < 1
False
```

✓ 조건문의 반환 값으로도 사용됨

```
>>> a = 5
>>> a < 6
True
>>> a > 6
False
```

❖ 자료형의 참과 거짓

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

- 문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있으면(" ", [ ], ( ), { }) 거짓이 됨
- 당연히 비어 있지 않으면 참이 됨
- 숫자는 그 값이 0일 때 거짓이 됨

- 참과 거짓이 프로그램에서 쓰이는 사례

```
>>> a = [1, 2, 3, 4]
>>> while a:          --- a가 참인 동안
...         a.pop()   --- 리스트의 마지막 요소를 하나씩 꺼낸다.
...
4
3
2
1
```

- ✓ 먼저 a = [1, 2, 3, 4]라는 리스트를 만듦
- ✓ while문은 조건문이 참인 동안 조건문 안에 있는 문장을 반복해서 수행함
- ✓ a가 참인 경우에 a.pop( )을 계속 실행하라는 의미임
- ✓ a.pop() 함수는 리스트 a의 마지막 요소를 끄집어 내고, 그 요소는 삭제함
- ✓ 결국 a가 빈 리스트([ ])가 되어 거짓이 되면, while 문에서 거짓이 되므로 중지됨

```
>>> if [ ]:                    --- 만약 [ ]가 참이면, 'True' 문자열 출력
...         print("True")
...     else:                   --- 만약 [ ]가 거짓이면, 'False' 문자열 출력
...         print("False")
...
False
```

✓ [ ]는 비어있는 리스트로 거짓이므로 False란 문자열이 출력됨

```
>>> if [1, 2, 3]:
...         print("True")
...     else:
...         print("False")
...
True
```

✓ [1, 2, 3]은 요소값이 있는 리스트로 참이므로 True를 출력함

❖ 불(bool) 연산

bool 내장 함수를 사용하면 자료형의 참과 거짓을 식별할 수 있음

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

✓ 'python' 문자열은 빈 문자열이 아니므로 bool 연산의 결과로 True를 돌려줌

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

✓ ' ' 문자열은 빈 문자열이므로 bool 연산의 결과로 False를 돌려줌

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

VIII. 자료형의 값을 저장하는 공간, 변수

❖ 변수를 만들 때는 =(assignment) 기호를 사용

```
변수명 = 변수에 저장할 값
```

- 변수를 만들 때 자료형을 직접 지정할 필요가 없음
- 변수에 저장된 값을 스스로 판단하여 자료형을 지정하기 때문에 더 편리함

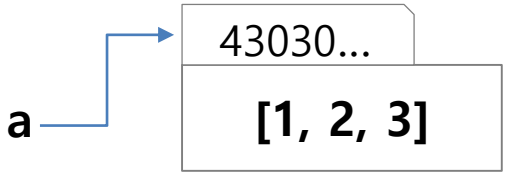
❖ 변수란?

- 파이썬에서 사용하는 변수는 객체를 가리키는 것

```
>>> 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 가리키게 됨

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



✓ **id()** 함수는 변수가 가리키고 있는 객체의 주소 값을 돌려줌

❖ 리스트를 복사할 때

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

✓ b라는 변수에 a가 가리키는 리스트를 대입하였음

```
>>> id(a)
4303029896
>>> id(b)
4303029896
>>> a is b          --- a와 b가 가리키는 객체는 동일한가?
True
```

✓ b는 a와 완전히 동일하다고 할 수 있음

✓ 다만 [1, 2, 3] 리스트를 참조하는 변수가 a변수 1개에서 b변수가 추가됨

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

- ✓ a 리스트의 두 번째 요소a[1]을 값 4로 바꾸었더니 b 리스트도 똑같이 바뀌었음
- ✓ 그 이유는 a, b 모두 같은 리스트를 가리키고 있기 때문임

- **b 변수를 생성할 때 a 변수의 값을 가져오면서 다른 주소를 가리키도록 만드는 방법**

1. [:] 이용

리스트 전체를 가리키는 [:]을 이용해서 복사함

```
>>> a = [1, 2, 3]
>>> b = a[:]     --- 리스트 a의 처음 요소부터 끝 요소까지 슬라이싱
>>> a[1] = 4
>>> a
[1, 4, 3]
>>> b
[1, 2, 3]
```

✓ a 리스트 값을 바꾸더라도 b 리스트에는 영향을 끼치지 않음

2. copy 모듈 이용

```
>>> from copy import copy
>>> a = [1, 2, 3]
>>> b = copy(a)
```

✓ b = copy(a)는 b = a[:]과 동일함

```
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>> b is a    --- b와 a가 가리키는 객체는 동일한가?
False
```

✓ 두 변수가 같은 값을 가지면서 다른 객체를 제대로 생성했는지 확인함

✓ b와 a가 가리키는 객체는 서로 다르다는 것을 알 수 있음

VIII. 자료형의 값을 저장하는 공간, 변수

❖ 변수를 만드는 여러 가지 방법

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

✓ 튜플로 a, b에 값을 대입할 수 있음

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

✓ 튜플은 괄호를 생략해도 됨

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

✓ 리스트로 변수를 만들 수 있음

```
>>> a = b = 'python'
```

✓ 여러 개의 변수에 같은 값을 대입할 수 있음

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

✓ a, b = b, a라는 문장을 수행한 후에는 그 값이 서로 바뀌었음을 확인할 수 있음

❖ 메모리에 생성된 변수 없애기

```
>>> a = 3
>>> b = 5
>>> del(a)
>>> del(b)
```

✓ 변수 a와 b가 3과 5 객체를 가리켰다가 del이라는 내장함수에 의해서 사라짐