자료형

강병준

숫자형 (Number)

1. 숫자형이란?

- 숫자형(Number)이란 숫자 형태로 이루어진 자료형으로, 우리가 이미 잘 알고 있는 것이다. 우리가 흔히 사용하는 것을 생각해 보자. 123 같은 정수, 12.34 같은 실수, 드물게 사용하긴 하지만 8진수나 16진수 같은 것도 있다.다음 표는 파이썬에서 숫자를 어떻게 사용하는지 간략하게 보여 준다.

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

숫자형은 어떻게 만들고 사용할까?

정수형

정수형(Integer)이란 말 그대로 정수를 뜻하는 자료형을 말한다. 다음 예는 양의 정수와 음의 정수, 숫자 0을 변수 a에 대입하는 예이다.

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

숫자형 (Number)

실수형

파이썬에서 실수형(Floating-point)은 소수점이 포함된 숫자를 말한다. 다음은 실수를 변수 a에 대입하는 예이다.

>>> a = 1.2; >> a = -3.45

위 방식은 우리가 일반적으로 볼 수 있는 실수형의 소수점 표현 방식이다.

>>> a = 4.24E10

>>> a = 4.24e-10

위 방식은 "컴퓨터식 지수 표현 방식"으로 파이썬에서는 4.24e10 또는 4.24E10처럼 표현한다(e와 E 둘 중 어느 것을 사용해도 무방하다).

8진수와 16진수

8진수(Octal)를 만들기 위해서는 숫자가 0o 또는 0O(숫자 0 + 알파벳 소문자 o 또는 대문자 O)로 시작하면 된다.

>>> a = 0o177

16진수(Hexadecimal)를 만들기 위해서는 0x로 시작하면 된다

>>> a = 0x8ff

>>> b = 0xABC

8진수나 16진수는 파이썬에서 잘 사용하지 않는 형태의 숫자 자료형이니 간단히 눈으로 익히고 넘어가자.

숫자형을 활용하기 위한 연산자

사칙연산

프로그래밍을 한 번도 해본 적이 없는 독자라도 사칙연산(+, -, *, /)은 알고 있을 것이다. 파이썬 역시 계산기와 마찬가지로 다음처럼 연산자를 사용해 사칙연산을 수행한다.

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

x의 y제곱을 나타내는 ** 연산자

다음으로 알아야 할 연산자로 **라는 연산자가 있다. 이 연산자는 x ** y처럼 사용했을 때 x의 y제곱(xy) 값을 돌려준다. 다음의 예를 통해 알아보자.

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

숫자형을 활용하기 위한 연산자

나눗셈 후 나머지를 반환하는 % 연산자

프로그래밍을 처음 접하는 독자라면 % 연산자는 본 적이 없을 것이다. %는 나눗셈의 나머지 값을 돌려주는 연산자이다. 7을 3으로 나누면 나머지는 1이 될 것이고 3을 7로 나누면 나머지는 3이 될 것이다. 다음 예로 확인해 보자.

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

나눗셈 후 몫을 반환하는 // 연산자

/ 연산자를 사용하여 7 나누기 4를 하면 그 결과는 예상대로 1.75가 된다.

```
>>> 7 / 4
1.75
```

이번에는 나눗셈 후 몫을 반환하는 // 연산자를 사용한 경우를 보자.

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

1.75에서 몫에 해당되는 정수값 1만 돌려주는 것을 확인할 수 있다.

숫자형 (Number)

```
print(4 + 7)        # 4 더하기 7
print(2 - 4)        # 2 빼기 4
print(5 * 3)        # 5 곱하기 3
print(7 % 3)        # 7을 3으로 나눈 나머지
print(2 ** 3)       # 2의 3제곱
print(2 + 3 * 2)    # 덧셈보다 곱셈을 먼저 계산
print(2 * (2 + 3))  # 괄호 안의 수식을 먼저 계산
```

```
11
-2
15
1
8
8
```

변수(variable)

▶ 변수란?

1. 변수의 사전적 의미 : 변화하는 것
2. 프로그래밍 언어에서 사용되는 변수
   값(value)이 저장된 메모리의 위치에 주어진 이름
   변수에 값을 배정(assignment)할 때 =기호를 사용
3. 프로그램에 전달되는 정보나 그 밖의 상황에 따라 바뀔 수 있는 값을 의미한다 즉 상수를 기억시킬 수 있는 기억 공간

▶ 변수 사용하기

어떤 값을 계속 사용할 때는 그 값을 변수(variable)에 담아두었다가 사용할 수 있다. 변수에 값을 넣는 것을 할당(assignment)한다고 이야기한다. 변수에 값을 할당할 때는 등호 기호를 사용하고 좌변에는 할당할 변수 이름을, 우변에는 할당할 값을 쓴다.

변수이름 = 변수값

변수의 이름은 알파벳으로 시작하며 뒤에는 숫자가 올 수 있다. 파이썬에서는 변수 이름의 대문자와 소문자를 구분하므로 주의하여야 한다. 즉, apple 과 Apple 과 APPLE 은 모두 서로 다른 변수이다.

숫자형 (Number)

정수형

1. 음의 정수, 0, 양의 정수
2. python에서는 메모리가 허용하는 한, 무한대의 정수를 다룰 수 있음.
3. 하지만 CPU 레지스터로 표현할 수 있는 크기보다 큰 정수를 다루는 경우 연산 속도는 느려집니다.
4. 일반적인 정수는 10진 정수로 간주
5. 0o로 시작하는 정수는 8진수
6. 0x, 0X로 시작하는 정수는 16진수
7. 0b로 시작하는 정수는 2진수
8. 문자열을 정수로 변환할 때는 int(문자열) 또는 int(문자열, 진수)를 이용
9. 문자열 대신에 실수를 대입하면 소수를 버립니다.
10. 실수로 된 문자열이나 복소수는 정수로 변환이 되지 않습니다.

수 다루기 - 정수

정수?

- 음의 정수, 0, 양의 정수
- 파이썬에서는 메모리가 허용하는 한, 무한대의 정수를 다룰 수 있음.

실습 1 (변수에 정수 입력)

```
>>> a = 3
>>> b = 123456789
>>> c = 12345678901234567890123456789012345678890
>>> a
3
>>> b
123456789
>>> c
12345678901234567890123456789012345678890
```

수 다루기 - 정수

실습 2 (변수에 음수 입력)

```
>>> d = -12345678901234567890123456789012345678901234567890 #
1234567890을 네 번 타이프
>>> e = -123456789
>>> f = -3
>>> d
-12345678901234567890123456789012345678901234567890
>>> e
-123456789
>>> f
-3
```

실습 3 (변수 형식 확인)

```
>>> f = -3
>>> type(f)
<class 'int'>
```

수 다루기 - 정수

파이썬에서 제공하는 사칙 연산결과

- 정수형은 컴퓨터 프로그래밍을 하면서 가장 많이 접할 자료형
- 파이썬의 규칙 상, 정수형과 정수형의 연산의 결과는 정수형
- 그러나 '나눗셈' 연산에는 예외가 적용됩니다.
- 정수형과 정수형 간의 연산이더라도, 결과값이 항상 소수형(floating point)으로 나오기 때문. 예를 들어 print(8 / 2)의 경우, 결과값이 소수형 4.0

실습 4 (덧셈과 뺄셈)

```
>>> a = 3 + 4
>>> a
7
>>> b = 7 - 10
>>> b
-3
```

수 다루기 - 정수

실습 5 (곱셈)

```
>>> c = 7 * -3
>>> c
-21
```

실습 6 (나눗셈의 몫 연산과 나머지 연산)

```
>>> d = 30 // 7
>>> d
4
>>> e = 30 % 7 # 30
>>> e
2
```

정수형

❖ 예제

```
a = 10
print("a =", a)
a = 0o12
print("a =", a)
a = 0xA
print("a =", a)
a = 0b1010
print("a =", a)
a = int('10')
print("a =", a)
a = int(10.8)
print("a =", a)
```

❖ 결과

```
a = 10
a = 10
a = 10
a = 10
a = 10
a = 10
```

실수형(소수형,Floating Point)

1. 파이썬에서는 실수를 지원하기 위해 부동 소수형을 제공
2. 부동 소수형은 소수점을 움직여서 소수를 표현하는 자료형
3. 부동 소수형은 8바이트만을 이용해서 수를 표현
4. 한정된 범위의 수만 표현
5. 디지털 방식으로 소수를 표현해야 하기 때문에 정밀도의 한계가 있음
6. 소수점을 표현하면 실수(1.2)
7. 지수(e)를 포함해도 실수(3e3, -0.2e-4)
8. 실수의 가장 큰 값과 작은 값은 sys 모듈의 float_info 또는 info.max, info.min으로 확인 가능
9. 무한대의 수는 float('inf' | '-inf')
10. is_integer 메소드를 통해서 정수로 변환시의 오차 문제 확인 가능
11. 실수를 정수로 변경할 때 사용할 때 사용할 수 있는 함수
12. int, round, math.ceil, math.floor

수 다루기 - 실수

파이썬에서는 실수를 지원하기 위해 부동 소수형을 제공

- “부동”은 뜰 부(浮), 움직일 동(動), 즉 떠서 움직인다는 뜻
- 부동 소수형은 소수점을 움직여서 소수를 표현하는 자료형

부동 소수형의 특징

- 부동 소수형은 8바이트만을 이용해서 수를 표현한다. 즉, 한정된 범위의 수만 표현할 수 있다.
- 디지털 방식으로 소수를 표현해야 하기 때문에 정밀도의 한계

실습 1

22/7의 결과는 무리수지만 부동소수형은 소수점 이하 15자리만 표현

```
>>> b = 22 / 7
>>> b
3.142857142857143
>>> type(b)
<class 'float'>
```

수 다루기 - 실수

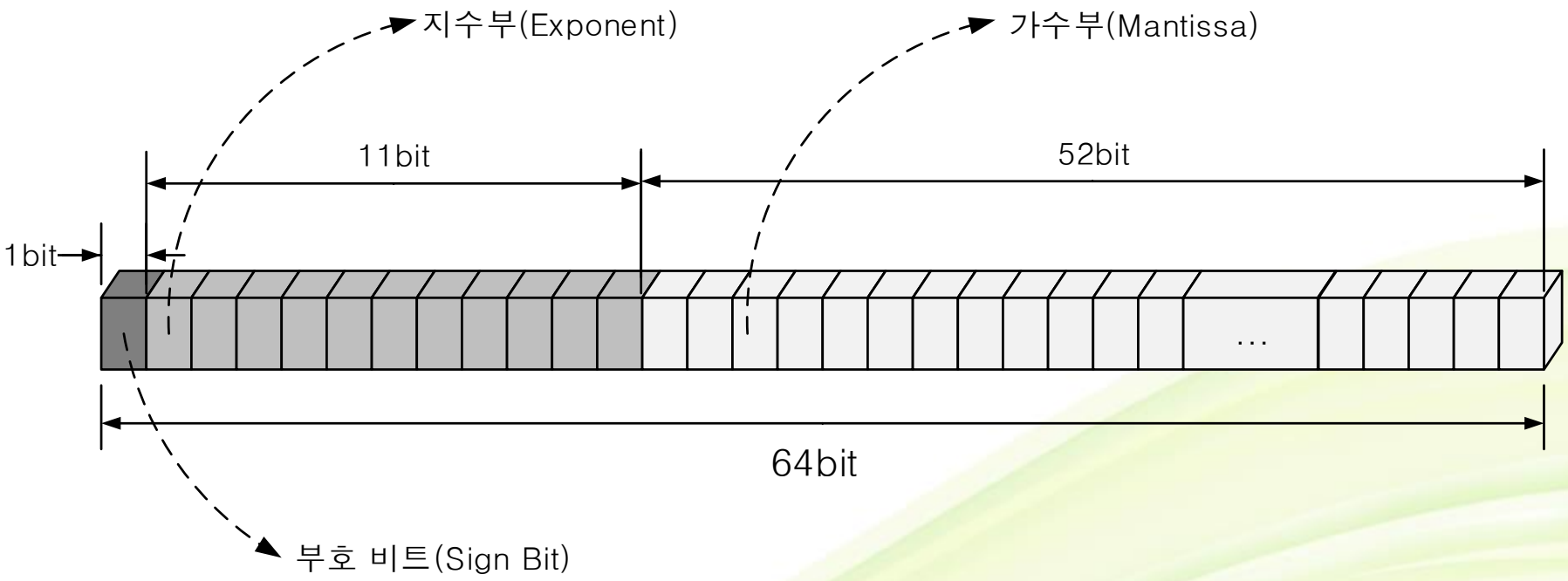
실습 2 (부동소수형의 사칙 연산)

```
>>> a = 1.23 + 0.32
>>> a
1.55
>>> b = 3.0 - 1.5
>>> b
1.5
>>> c = 2.1 * 2.0
>>> c
4.2
>>> d = 4.5 // 2.0
>>> d
2.0
>>> e = 4.5 % 2.0
>>> e
0.5
>>> f = 4.5 / 2.0
>>> f
2.25
```

수 다루기 - 실수

❖부동 소수형의 구조

- IEEE 754



복소수

복소수

1. 실수부와 허수부로 숫자를 표현
2. 허수부는 j를 추가해서 표현
3. 복소수에서 real 속성을 호출하면 실수부만 리턴
4. imag를 호출하면 허수부만 리턴
5. 정수 2개를 가지고 복소수 생성 가능 – complex(실수부, 허수부)
6. conjugate()를 호출해서 켤레 복소수 리턴

```
c = 3 + 4j
print(c)
a = 3;
b = 4;
print(complex(3,4))
d = c.conjugate();
print(d)
```

분수 표현

fractions 모듈을 이용한 분수 표현

- ✓ fractions 모듈을 이용해서 분수 표현 가능
- ✓ 5/7을 표현하고자 할 때 Fraction('5/7') 또는 Fraction(5,7)
- ✓ 그리고 만들어진 데이터의 numeratior 와 denominator를 이용해서 분자와 분모를 따라 추출 가능

```
from fractions import Fraction
a = Fraction(5,7) + Fraction('2/5')
print(a)
```

연습문제

1. 파이썬 연산

3×2−8÷4

25×6÷3+17

39021−276920÷12040

26−10%6

2. 연산순서와 괄호

괄호가 있으면 괄호 안을 먼저 계산한다. 하지만 파이썬은 소괄호, 중괄호, 대괄호를 구분하지 않고 모두 소괄호 기호를 사용한다. 예를 들어 아래 수식을 파이썬 코드로 나타내면 다음과 같다.

12−(5×7+1)

5×(8+(10−6)÷2)

48320−((365−5×9)÷16)×987

((34−3×7)%5+4)2

문자열이란?

문자열(String)이란 문자, 단어 등으로 구성된 문자들의 집합을 의미한다. 예를 들어 다음과 같은 것들이 문자열이다.

"Life is too short, You need Python"; "a"

"123"

위 문자열 예문을 보면 모두 큰따옴표(" ")로 둘러싸여 있다. "123은 숫자인데 왜 문자열이지?"라는 의문이 드는 독자도 있을 것이다. 따옴표로 둘러싸여 있으면 모두 문자열이라고 보면 된다.

문자열은 어떻게 만들고 사용할까?

위 예에서는 문자열을 만들 때 큰따옴표(" ")만을 사용했지만 이 외에도 문자열을 만드는 방법은 3가지가 더 있다. 파이썬에서 문자열을 만드는 방법은 총 4가지이다.

1. 큰따옴표(")로 양쪽 둘러싸기

"Hello World"

2. 작은따옴표(')로 양쪽 둘러싸기

'Python is fun'

3. 큰따옴표 3개를 연속(""")으로 써서 양쪽 둘러싸기

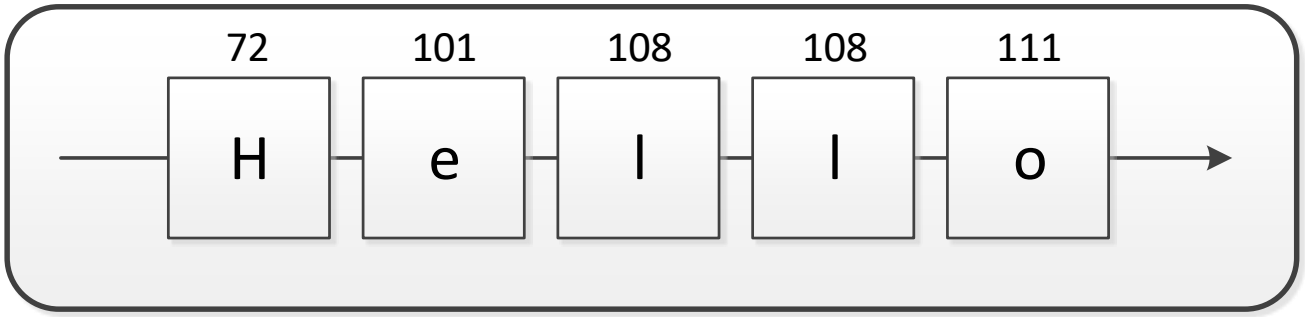
"""Life is too short, You need python"""

4. 작은따옴표 3개를 연속(''')으로 써서 양쪽 둘러싸기

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

텍스트 다루기

- 프로그래밍 언어마다 방식이 조금씩 다르긴 하지만 대부분은 다음 그림과 같이 개별 문자를 나타내는 수를 이어서 텍스트를 표현



- 파이썬에서는 텍스트를 다루는 자료형으로 string을 제공
  - string은 영어로 끈, 줄 등의 뜻을 갖고 있으므로 문자를 끈으로 가지런히 묶어놓은 것이라고 이해
  - 우리 말로는 문자열 → 문자열(文字列)도 문자를 가지런히 늘어놨다는 뜻

문자열

1. 파이썬의 문자열
2. 한 줄 문자열: ‘’ 또는 “” 안에 기재
3. 여러 줄 문자열:‘’’ ’’’ 또는 “”” “”” 안에 기재
4. escape 문자(₩를 이용해서 표현)
    - ₩enter: 다음 줄과 연속
    - ₩₩: ₩문자
    - ₩’, ₩”: 작은 따옴표나 큰 따옴표
    - ₩b: 백스페이스
    - ₩n 또는 ₩012: 줄 변경
    - ₩t: 탭

문자열

문자열 안에 작은따옴표나 큰따옴표를 포함시키고 싶을 때

문자열을 만들어 주는 주인공은 작은따옴표(')와 큰따옴표(")이다. 그런데 문자열 안에도 작은따옴표와 큰따옴표가 들어 있어야 할 경우가 있다. 이때는 좀 더 특별한 기술이 필요하다. 예제를 하나씩 살펴보면서 원리를 익혀 보자.

1. 문자열에 작은따옴표 (') 포함시키기

Python's favorite food is perl

위와 같은 문자열을 food 변수에 저장하고 싶다고 가정하자. 문자열 중 Python's에 작은따옴표(')가 포함되어 있다.

이럴 때는 다음과 같이 문자열을 큰따옴표(")로 둘러싸야 한다. 큰따옴표 안에 들어 있는 작은따옴표는 문자열을 나타내기 위한 기호로 인식되지 않는다.

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

프롬프트에 food를 입력해서 결과를 확인하자. 변수에 저장된 문자열이 그대로 출력되는 것을 볼 수 있다.

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

시험 삼아 다음과 같이 큰따옴표(")가 아닌 작은따옴표(')로 문자열을 둘러싼 후 다시 실행해 보자. 'Python'이 문자열로 인식되어 구문 오류(SyntaxError)가 발생할 것이다.

문자열

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

2. 문자열에 큰따옴표 (") 포함시키기

"Python is very easy." he says.

위와 같이 큰따옴표(")가 포함된 문자열이라면 어떻게 해야 큰따옴표가 제대로 표현될까? 다음과 같이 문자열을 작은따옴표(')로 둘러싸면 된다.

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

이렇게 작은따옴표(') 안에 사용된 큰따옴표(")는 문자열을 만드는 기호로 인식되지 않는다.

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

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

작은따옴표(')나 큰따옴표(")를 문자열에 포함시키는 또 다른 방법은 백슬래시(₩)를 사용하는 것이다. 즉 백슬래시(₩)를 작은따옴표(')나 큰따옴표(") 앞에 삽입하면 백슬래시(₩) 뒤의 작은따옴표(')나 큰따옴표(")는 문자열을 둘러싸는 기호의 의미가 아니라 문자 ('), (") 그 자체를 뜻하게 된다.

문자열

여러 줄인 문자열을 변수에 대입하고 싶을 때

문자열이 항상 한 줄짜리만 있는 것은 아니다. 다음과 같이 여러 줄의 문자열을 변수에 대입하려면 어떻게 처리해야 할까?

Life is too short

You need python

1. 줄을 바꾸기 위한 이스케이프 코드 ₩n 삽입하기

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

줄바꿈 문자 ₩n을 삽입하는 방법이 있지만 읽기에 불편하고 줄이 길어지는 단점이 있다.

2. 연속된 작은따옴표 3개(''') 또는 큰따옴표 3개(""") 사용하기

위 1번의 단점을 극복하기 위해 파이썬에서는 다음과 같이 작은따옴표 3개(''') 또는 큰따옴표 3개(""")를 사용한다.

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

작은따옴표 3개를 사용한 경우

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

큰따옴표 3개를 사용한 경우 print(multiline)을 입력해서 어떻게 출력되는지 확인해 보자.

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

문자열 연산

덧셈 연산은 두 문자열을 붙이고 곱셈 연산은 문자열을 반복한다.

```
print("내 이름은 " + "홍길동" + "입니다.")
```

내 이름은 홍길동입니다.

```
print("*" * 10)
```

****

한 줄 띄우기

print 명령은 한 번 호출할 때마다 한 줄씩 출력한다. 만약 print 명령을 한 번만 쓰면서 여러 줄에 걸쳐 출력을 하고 싶으면 문자열에 "다음 줄 넘기기(line feed) 기호"인 ₩n를 넣어야 한다.

```
print("한 줄 쓰고₩n그 다음 줄을 쓴다.")
```

한 줄 쓰고

그 다음 줄을 쓴다.

줄을 바꾸지 않고 이어서 출력하기

반대로 print 명령을 여러 번 쓰면서 줄은 바꾸지 않고 싶다면 다음과 같이 print 명령에 end=""이라는 인수를 추가한다.

```
print("한 줄 쓰고 ", end="")
print("이어서 쓴다.")
```

한 줄 쓰고 이어서 쓴다.

문자열 인덱싱과 슬라이싱

인덱싱(Indexing)이란 무엇인가를 "가리킨다"는 의미이고, 슬라이싱(Slicing)은 무엇인가를 "잘라낸다"는 의미이다. 이런 의미를 생각하면서 다음 내용을 살펴보자.

```
>>> a = "Life is too short, You need Python"
Life is too short, You need Python
0         1         2         3
01234567890123456789012345678901234567890123
```

"Life is too short, You need Python" 문자열에서 L은 첫 번째 자리를 뜻하는 숫자 0, 바로 다음인 i는 1 이런 식으로 계속 번호를 붙인 것이다. 중간에 있는 short의 s는 12가 된다.

이제 다음 예를 실행해 보자.

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

a[3]이 뜻하는 것은 a라는 문자열의 네 번째 문자 e를 말한다.

파이썬은 0부터 숫자를 센다."

따라서 파이썬은 위 문자열을 다음과 같이 바라보고 있다.

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

0부터 숫자를 센다는 것이 처음에는 익숙하지 않겠지만 계속 사용하다 보면 자연스러워질 것이다. 위 예에서 볼 수 있듯이 a[번호]는 문자열 안의 특정한 값을 뽑아내는 역할을 한다. 이러한 작업을 인덱싱이라고 한다.

문자열 인덱싱 활용하기

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

앞의 a[0]과 a[12]는 쉽게 이해할 수 있는데 마지막의 a[-1]이 뜻하는 것은 뭘까? 눈치 빠른 독자는 이미 알아챘겠지만 문자열을 뒤에서부터 읽기 위해 마이너스(-) 기호를 붙이는 것이다. 뒤에서부터 첫 번째 문자를 표시할 때도 0부터 세어 "a[-0]이라고 해야 하지 않을까?"라는 의문이 들 수도 있겠지만 잘 생각해 보자. 0과 -0은 똑같은 것이기 때문에 a[-0]은 a[0]과 똑같은 값을 보여 준다.

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

계속해서 몇 가지 예를 더 보자.

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

위 첫 번째 예는 뒤에서부터 두 번째 문자를 가리키는 것이고, 두 번째 예는 뒤에서부터 다섯 번째 문자를 가리키는 것이다.

문자열 슬라이싱이란?

그렇다면 "Life is too short, You need Python" 문자열에서 단순히 한 문자만을 뽑아내는 것이 아니라 'Life' 또는 'You' 같은 단어를 뽑아내는 방법은 없을까?

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

※ 인덱싱 기법과 슬라이싱 기법은 뒤에서 배울 자료형인 리스트나 튜플에서도 사용할 수 있다.

위 예는 슬라이싱 기법으로 다음과 같이 간단하게 처리할 수 있다.

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

a[0:4]가 뜻하는 것은 a 문자열, 즉 "Life is too short, You need Python" 문장에서 자리 번호 0부터 4까지의 문자를 뽑아낸다는 뜻이다>>> a[0:3]   'Lif'

이렇게 되는 이유는 간단하다. 슬라이싱 기법으로 a[시작 번호:끝 번호]를 지정할 때 끝 번호에 해당하는 것은 포함하지 않기 때문이다. a[0:3]을 수식으로 나타내면 다음과 같다.

```
0 <= a < 3
```

이 수식을 만족하는 것은 a[0], a[1], a[2]이다. 따라서 a[0:3]은 'Lif'이고 a[0:4]는 'Life'가 되는 것이다. 이 부분이 문자열 연산에서 가장 혼동하기 쉬운 부분이니 장 마지막의 연습 문제를 많이 풀어 보면서 몸에 익히기 바란다

문자열을 슬라이싱하는 방법

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

위 예는 a[0] + a[1] + a[2] + a[3] + a[4]와 동일하다. a[4]는 공백 문자이기 때문에 'Life'가 아닌 'Life '가 출력된다.

>>> a[0:2] 'Li'

>>> a[5:7] 'is'

>>> a[12:17] 'short'

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

>>> a[19:]

'You need Python'

a[시작 번호:끝 번호]에서 시작 번호를 생략하면 문자열의 처음부터 끝 번호까지

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

a[시작 번호:끝 번호]에서 시작 번호와 끝 번호를 생략하면 문자열의 처음부터 끝

>>> a[:]

'Life is too short, You need Python'

슬라이싱에서도 인덱싱과 마찬가지로 마이너스(-) 기호를 사용할 수 있다.

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

위 소스 코드에서 a[19:-7]이 뜻하는 것은 a[19]에서부터 a[-8]까지를 말한다. 이 역시 a[-7]은 포함하지 않는다.

슬라이싱으로 문자열 나누기

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

위 예는 문자열 a를 두 부분으로 나누는 기법이다. 숫자 8을 기준으로 문자열 a를 양쪽으로 한 번씩 슬라이싱했다. a[:8]은 a[8]이 포함되지 않고, a[8:]은 a[8]을 포함하기 때문에 8을 기준으로 해서 두 부분으로 나눌 수 있는 것이다. 위 문자열 "20010331Rainy"를 연도 2001, 월과 일을 나타내는 0331, 날씨를 나타내는 Rainy의 세 부분으로 나누려면 다음과 같이 할 수 있다.

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

숫자 4와 8로 "20010331Rainy" 문자열을 세 부분으로 나누는 방법을 보여 준다.

연습문제

["Pithon"이라는 문자열을 "Python"으로 바꾸려면?]

Pithon 문자열을 Python으로 바꾸려면 어떻게 해야 할까? 제일 먼저 떠오르는 생각은 다음과 같을 것이다.

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

즉 a 변수에 "Pithon" 문자열을 대입하고 a[1]의 값이 i니까 a[1]을 y로 바꾸어 준다는 생각이다. 하지만 결과는 어떻게 나올까?

당연히 오류가 발생한다. 왜냐하면 문자열의 요소값은 바꿀 수 있는 값이 아니기 때문이다(문자열 자료형은 그 요소값을 변경할 수 없다. 그래서 immutable한 자료형이라고도 부른다).

슬라이싱 기법을 사용하면 Pithon 문자열을 사용해 Python 문자열을 만들 수 있다.

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

위 예에서 볼 수 있듯이 슬라이싱을 사용하면 "Pithon" 문자열을 'P' 부분과 'thon' 부분으로 나눌 수 있기 때문에 그 사이에 'y' 문자를 추가하여 'Python'이라는 새로운 문자열을 만들 수 있다.

문자열

문자열은 근본적으로 데이터를 변경할 수 없음
문자열 내의 특정 문자의 값을 변경할 수 없습니다.

str = “hello”
str[0] = ‘f’=> 이런 문장은 에러

문자열을 추가하거나 삭제할 때는 슬라이싱과 연결하기를 이용해야 합니다.

- Hardward and Shell 문자열에서 and 앞에 Kernel을 추가하기

```
str = "Hardware and Shell"
str = str[:8] + ' Kernel ' + str[8:]
print(str)
```

- 위의 문자열에서 and 제거하기

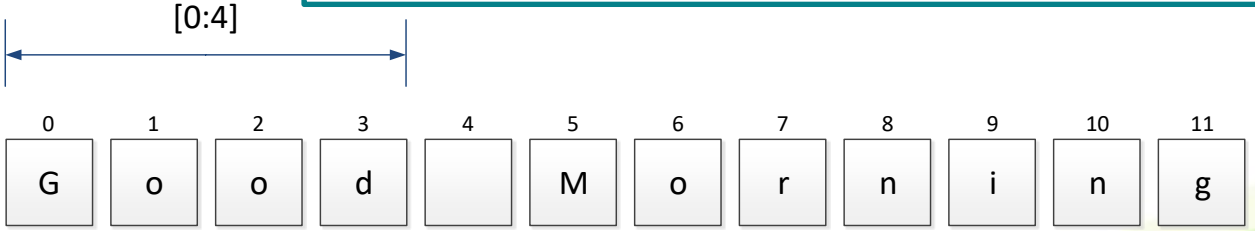
```
str = str[:15] + str[20:]
print(str)
```

텍스트 다루기

- 문자열 분리(슬라이싱 Slicing)는 [와] 연산자를 통해 수행함
- 실습 3 (문자열 분리)

```
>>> s = 'Good Morning'
>>> s[0:4]
'Good'
```

문자열 S의 0번째 문자부터 4번째 문자 앞까지를 분리합니다.



- 슬라이싱은 문자열 뿐 아니라 다른 순서열 자료형에서도 사용 가능
- 문자열이든 순서열이든 슬라이싱을 하더라도 원본은 그대로 유지
- 실습 4 (문자열 분리2)

```
>>> a = 'Good Morning'
>>> b = a[0:4]
>>> c = a[5:12]
>>> a
'Good Morning'
>>> b
'Good'
>>> c
'Morning'
```

문자열 a를 슬라이싱해서 b를 만들어내도 a는 여전히 'Good Morning'입니다. 따라서 c를 만들어 낼 때도 a의 원본 그대로가 사용됩니다.

텍스트 다루기

- 특정 위치에 있는 문자를 참조하고 싶을 때는 대괄호 [와] 사이에 사이에 첨자(Index) 번호 하나만 입력
- 실습 5

```
>>> a = 'Good Morning'
>>> a[0]
'G'
>>> a[8]
'n'
```

- **in** 연산자는 프로그래머가 원하는 부분이 문자열 안에 존재하는지를 확인
- 실습 6

```
>>> a = 'Good Morning'
>>> 'Good' in a
True
>>> 'X' in a
False
>>> 'Evening' in a
False
```

텍스트 다루기

- len() : 순서열 길이를 재는 함수. 문자열에도 사용 가능.
- 실습 7 (len())

```
>>> a = 'Good Morning'
>>> len(a)
12
```

숫자형에서의 *는 왼쪽의 수와 오른쪽의 수를 '수학적으로 곱하라'는 의미였죠?

하지만 문자열에서의 *는 왼쪽의 문자열을 오른쪽의 수만큼 '반복하라'는 의미

```
fly = "날아라! "

print("떴다떴다 비행기!")

print(fly * 2)
```

```
떴다떴다 비행기!

날아라! 날아라!
```

텍스트 다루기 - 문자열 메소드

메소드	설명

startswith()	원본 문자열이 매개변수로 입력한 문자열로 시작되는지를 판단합니다. 결과는 True 또는 False로 나옵니다.
	>>> a = 'Hello'
	>>> a.startswith('He')
	True
	>>> a.startswith('lo')
	False
	>>>

endswith()	원본 문자열이 매개변수로 입력한 문자열로 끝나는지를 판단합니다.
	결과는 True 또는 False로 나옵니다.
	>>> a = 'Hello'
	>>> a.endswith('He')
	False
	>>> a.endswith('lo')
	True

find()	원본 문자열 안에 매개변수로 입력한 문자열이 존재하는 위치를 앞에서부터 찾습니다.
	존재하지 않으면 -1을 결과로 내놓습니다.
	
	>>> a = 'Hello'
	>>> a.find('ll')
	2
	>>> a.find('H')
	0
	>>> a.find('K')
	-1

텍스트 다루기 - 문자열 메소드

메소드	설명

rfind()	원본 문자열 안에 매개변수로 입력한 문자열이 존재하는 위치를 뒤에서부터 찾습니다. 존재하지 않으면 -1을 결과로 내놓습니다.
	>>> a = 'Hello'
	>>> a.rfind('H')
	0
	>>> a.rfind('lo')
	3
	>>> a.rfind('M')
	-1

count()	원본 문자열 안에 매개변수로 입력한 문자열이 몇 번 등장하는지를 셉니다.
	>>> a = 'Hello'
	>>> a.count('l')
	2

lstrip()	원본 문자열 왼쪽에 있는 공백을 제거합니다. >>> ' Left Strip'.lstrip() 'Left Strip'
rstrip()	원본 문자열 오른쪽에 있는 공백을 제거합니다. >>> 'Right Strip '.rstrip() 'Right Strip'
strip()	원본 문자열 양쪽에 있는 공백을 제거합니다. >>> ' Strip '.strip() 'Strip'

텍스트 다루기 - 문자열 메소드

메소드	설명

isalpha()	원본 문자열이 숫자와 기호를 제외한 알파벳(영문, 한글 등)으로만 이루어져 있는지를 평가합니다.
	>>> 'ABCDefgh'.isalpha()
	True
	>>> '123ABC'.isalpha()
	False

isnumeric()	원본 문자열이 수로만 이루어져 있는지를 평가합니다.
	>>> '1234'.isnumeric()
	True
	>>> '123ABC'.isnumeric()
	False

isalnum()	원본 문자열이 알파벳과 수로만 이루어져 있는지를 평가합니다.
	>>> '1234ABC'.isalnum()
	True
	>>> '1234'.isalnum()
	True
	>>> 'ABC'.isalnum()
	True
	>>> '1234 ABC'.isalnum()
	False

텍스트 다루기 - 문자열 메소드

메소드	설명

replace()	원본 문자열에서 찾고자 하는 문자열을 바꾸고자 하는 문자열로 변경합니다.
	>>> a = 'Hello, World'
	>>> b = a.replace('World', 'Korea')
	>>> a
	'Hello, World'
	>>> b
	'Hello, Korea'

split()	매개변수로 입력한 문자열을 기준으로 원본 문자열을 나누어 리스트를 만듭니다. 리스트는 목록을 다루는 자료형이며 다음 장에서 자세히 다룹니다.
	>>> a = 'Apple, Orange, Kiwi'
	>>> b = a.split(',')
	>>> b
	['Apple', ' Orange', ' Kiwi']
	>>> type(b)
	<class 'list'>

upper()	원본 문자열을 모두 대문자로 바꾼 문자열을 내놓습니다.
	>>> a = 'lower case'
	>>> b = a.upper()
	>>> a
	'lower case'
	>>> b
	'LOWER CASE'

텍스트 다루기 - 문자열 메소드

메소드	설명

lower()	원본 문자열을 모두 소문자로 바꾼 문자열을 내놓습니다.
	
	>>> a = 'UPPER CASE'
	>>> b = a.lower()
	>>> a
	'UPPER CASE'
	>>> b
	'upper case'

format()	형식을 갖춘 문자열을 만들 때 사용합니다. 문자열 안에 중괄호 {와 }로 다른 데이터가 들어갈 자리를 만들어 두고 format() 함수를 호출할 때 이 자리에 들어갈 데이터를 순서대로 넣어주면 원하는 형식의 문자열을 만들어 낼 수 있습니다.
	
	>>> a = 'My name is {0}. I am {1} years old.'.format('Mario', 40)
	>>> a
	'My name is Mario. I am 40 years old.'
	>>> b = 'My name is {name}. I am {age} years old.'.format(name='Luigi', age=35)
	>>> b
	'My name is Luigi. I am 35 years old.'

텍스트 다루기 - 문자열 메소드

문자열 삽입(join)

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

abcd 문자열의 각각의 문자 사이에 ','를 삽입한다.

join 함수는 문자열뿐만 아니라 앞으로 배울 리스트나 튜플도 입력으로 사용할 수 있다(리스트와 튜플은 곧 배울 내용이니 여기에서는 잠시 눈으로만 살펴보자). join 함수의 입력으로 리스트를 사용하는 예는 다음과 같다.

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

형 변환

```
# 1. 소수형에서 정수형으로
print(int(3.8))
# 2. 정수형에서 소수형으로
print(float(3))
# 3. 문자열에서 정수형으로
print(int("2") + int("5"))
# 4. 문자열에서 소수형으로
print(float("1.1") + float("2.5"))
# 5. 정수형에서 문자열로
print(str(2) + str(5))
# 6. 정수형에서 문자열로
print("제 나이는 " + str(7) + "살입니다.")
```

```
3
3.0
7
3.6
25
제 나이는 7살입니다.
```

형 변환

int는 괄호 안의 값을 정수형으로 변환해줍니다.
int(3.8)의 경우 소수 부분인 .8을 버려지면서 정수 부분인 3만 남게 됩니다.

float는 괄호 안의 값을 소수형으로 변환해줍니다.
float(3)의 경우 정수 3에 소수 부분이 추가되어 3.0이 됩니다.

이 경우 int는 쌍따옴표를 없애서 괄호 안의 문자열을 정수형으로 변환해줍니다.
int("2")는 정수형 2, int("5")는 정수형 5가, int("2") + int("5")는 정수형 7이 됩니다.

이 경우 float는 쌍따옴표를 없애서 괄호 안의 문자열을 소수형으로 변환
float("1.1")은 소수형 1.1이 되고, float("2.5")는 소수형 2.5이 되어서, float("1.1") + float(2.5)는 소수형 3.6이 됩니다.

str은 괄호 안의 값을 문자열로 변환해줍니다. str(2)는 정수형 2를 문자열 "2"로 바꿔주고, str(5)는 정수형 5를 문자열 "5"로 바꿔줘서, str(2) + str(5)는 문자열 "2"와 문자열 "5"를 이어 연결해준 문자열 "25"가 됩니다.

str(7)은 문자열 "7"이기 때문에, 문자열 덧셈 연산에 의해 "제 나이는 7살입니다."

수에서 텍스트로, 텍스트에서 수로

- 이 코드는 '정상적으로'동작할까?

```
a = input()
b = input()
result = a * b
```

a와 b는 문자열입니다. 이 코드는 파이썬이 수행할 수 없습니다.

  - input() 함수의 결과는 문자열이므로 * 연산자를 사용할 수 없음.

- 문자열을 숫자로 바꾸기 위해서는 int(), float(), complex()를 사용
- 실습 1

```
>>> int('1234567890')
1234567890
>>> float('123.4567')
123.4567
>>> complex('1+2j')
(1+2j)
```

수에서 텍스트로, 텍스트에서 수로

❖ 실습 2 : *04/input_multiply.py*

```
print("첫 번째 수를 입력하세요. : ")
a = input()
print("두 번째 수를 입력하세요. : ")
b = input()

result = int(a) * int(b)

print("{0} * {1} = {2}".format(a, b, result))
```

- 실행 결과

```
>input_multiply.py
첫 번째 수를 입력하세요. :
5
두 번째 수를 입력하세요. :
4
5 * 4 = 20
```

수에서 텍스트로, 텍스트에서 수로

- 숫자를 문자열로 바꾸기 위해서는 str()을 사용
- 실습 3

```
>>> import math
>>> type(math.pi)
<class 'float'>
>>> text = "원주율은 " + str(math.pi) + "입니다."
>>> text
'원주율은 3.141592653589793입니다.'
```

문자열

문자열의 서식 지정

1. print() 함수를 사용하면 출력 문자열의 서식을 자유롭게 지정 가능
2. 단순하게 문자열이나 데이터를 , 를 이용해서 출력할 수 있지만 양식을 만들어두고 빈칸에 필요한 내용을 채워서 문서를 자유롭게 채워나갈 수 있음
3. 이전 방식은 문자열 안에 서식 문자를 지정하고 튜플을 이용해서 값을 채움
4. 서식 문자는 %와 함께 표현

```
int_val = 23
float_val =  45.9876
print("int_val:%3d, float_val:%0.2f" % (int_val, float_val))
```

1. 파이썬 3.0에서는 (데이터, '서식')을 이용해서 서식을 지정하기를 권장

```
int_val = 23format
float_val =  45.9876
print("int_val:", format(int_val,"3d"), "float_val:", format(float_val, "0.2f"))
```

문자열

문자열의 서식 지정

1. 문자열의 format 메소드를 이용
2. 문자열 안에 {}를 사용하고 .format(데이터 나열)을 이용하면 나열된 데이터가 순차적으로 대입
3. {인덱스}를 이용하면 format에서 인덱스 번째 데이터가 대입
4. {인덱스:서식}을 이용하면 인덱스 번째의 데이터를 서식에 맞추어 출력

```
str = [1,3,4]
print('최대값:{0:5d}₩t최소값:{1:5d}'.format(max(str), min(str)))
```

✓ 서식 자료형
- d는 정수, f는 실수, s는 문자열

✓ 서식에 사용할 수 있는 보조 서식 문자
- 숫자: 숫자만큼의 최소자리 확보
- 숫자.숫자f: 앞의 숫자만큼 자리를 확보하고 뒤의 숫자만큼 소수 표현
- <: 왼쪽 맞춤
- >: 오른쪽 맞춤
- 공백: 양수 일 때 앞에 공백 출력
- +: 양수 일 때 + 출력
- 0: 왼쪽의 빈공간을 0으로 채움

문자열

예제

```
str = "c:₩₩data₩₩data.txt"

#test의 존재여부

print(str.find('test'))

#위의 문자열에서 파일의 확장자만 추출하기

list = str.split(".")

print("확장자:", list[len(list)-1])

print(str.replace('data', 'python'))
```

문자열

탭 문자 변환 메소드

1. expandtabs(숫자): 숫자를 생략하면 탭을 8자 공백으로 변경하고 숫자를 대입하면 숫자만큼의 공백으로 변경

문자열 확인 메소드

1. isdigit()
2. isnumeric()
3. isdecimal()
4. isalpha()
5. isalnum()
6. islower()
7. isupper()
8. isspace()
9. istitle()
10. isidentifier()
11. isprintable()

문자열

문자열 매핑 메소드

1. maketrans()와 translate()를 이용하면 문자열 매핑 가능
2. maketrans()를 이용해서 치환할 문자열을 만들고 translate의 매개변수로 대입

```
instr = 'abcdef'
outstr = '123456'
trans = ''.maketrans(instr, outstr)
str = 'hello world'
print(str.translate(trans))
```

문자열

문자열 인코딩

1. 파이썬의 문자열은 기본적으로 utf-8 인코딩을 사용하는데 기본 아스키 코드는 1바이트를 사용하고 나머지는 2바이트를 사용하는 방식의 인코딩입니다.
2. 문자열을 바이트 코드로 변환할 때는 encode()를 이용하고 특정 코드로 변환하고자 하는 경우에는 encode('인코딩방식') 메소드를 이용합니다.
3. 바이트 코드는 문자열 함수를 거의 사용이 가능하지만 문자열과 직접 연산은 불가능합니다.
4. 바이트 코드를 문자열로 변환할 때는 decode('인코딩방식')을 이용해서 문자열로 변환할 수 있습니다.
5. ord('문자'):문자를 코드 값으로 변환
6. chr(유니코드): 유니코드를 문자로 변환

```
b = '파이썬'.encode('utf-8')
print(b)
b = '파이썬'.encode('ms949')
print(b)
str = b.decode('ms949') #utf-8로 인코딩 하면 예외 발생
print(str)
```

자료형 퀴즈

(1) 다음 프로그램은 어떤 값을 출력할까요**?**

```
print(10 / (10 % 6))
```

1) 2
2) 2.0
3) 2.5
4) 10
5) 10.0

(해설)
괄호가 포함된 수식의 경우 괄호 안부터 계산한 후 괄호 밖과 연산해주어야 합니다.
괄호 안의 10 % 6은 10을 6으로 나눈 나머지라는 뜻입니다.
10을 6으로 나누면 몫이 1이고 나머지가 4이기 때문에, 괄호 안의 값은 4입니다.
10을 괄호 안의 4로 나누면 2.5가 나오겠죠?
따라서 답은 (3)번입니다.

자료형 퀴즈

(2) 실행했을 때 콘솔에 8.0이 출력되는 프로그램을 모두 고르세요.

1) print(2 ** 3.0)
2) print(int("3") + float("5"))
3) print(str(4.0) * 2)
4) print(float(int(42 / 5)))
5) print(2 * (3 + 1))

(해설)

(1) 2의 3승은 8이지만, 2 ** 3.0은 정수형과 소수형 간의 연산이기 때문에 소수형 8.0이 나옵니다.
(2) int("3")은 정수형 3이 되고, float("5")는 소수형 5.0이 됩니다. 3 + 5.0은 정수형과 소수형 간의 연산이기 때문에 소수형 8.0이 나옵니다.
(3) str(4.0)은 문자열 "4.0"이 되고, "4.0" * 2는 문자열 "4.0"을 두 번 반복하라는 뜻입니다. "4.0" + "4.0"과 같기 때문에 "4.04.0"이 나옵니다.
(4) 42를 5로 나누면 8.4가 나옵니다. int(8.4)는 8.4의 소수 부분을 빼서 8이 되고, float(8)은 8에 소수 부분을 붙여서 8.0이 나옵니다.
(5) 괄호 안의 3 + 1은 4가 되고, 2 * 4는 정수형 8이 나옵니다.
파이썬에서 정수형 8과 소수형 8.0은 비슷하지만 다른 값입니다.
따라서 (1), (2), (4)번이 정답입니다.

자료형 퀴즈

print문을 이용하여, 다음의 두 문장을 출력하세요.

1) '응답하라 1988'은 많은 시청자들에게 사랑을 받은 드라마에요.

2) 데카르트는 "나는 생각한다. 고로 존재한다."라고 말했다

힌트 : print("맥도날드 버거 중에 나는 '빅맥'을 제일 좋아한다.")

힌트 : print('스티브잡스는 "Stay Hungry. Stay Foolish."라고 말했다.')

문자열/형 변환 연습

1) 형 변환과 2) 문자열 덧셈 연산을 이용하여, 아래의 내용을 출력하세요.
여러분은 과제를 하시고 있는 현재 날짜와 요일을 출력하시면 됩니다.

오늘은 2016년 12월 25일 일요일입니다.

힌트 : 수 노래방의 1시간 가격은 20000원입니다.를 출력하기 위해서는 아래와 같이 하시면 됩니다.

```
print("수 노래방의 " + str(1) + "시간 가격은 " + str(20000) + "원입니다.")
```

```
print("오늘  오후 " + str(10) + "시의 온도는 " + str(21) + "도이다.")
```

```
year = 2016
month = 1
day = 16
day_of_week = "일"
```

```
print("오늘은 " + str(year) + "년 " + str(month) + "월 " + str(day) + "일 " + day_of_week + "요일입니다.")
```

오늘은 2016년 1월 16일 일요일입니다.

문자열 포매팅

문자열에서 또 하나 알아야 할 것으로는 문자열 포매팅(Formatting)이 있다. 이것을 공부하기 전에 다음과 같은 문자열을 출력하는 프로그램을 작성했다고 가정해 보자.

"현재 온도는 18도입니다."

시간이 지나서 20도가 되면 다음 문장을 출력한다.

"현재 온도는 20도입니다"

위 두 문자열은 모두 같은데 20이라는 숫자와 18이라는 숫자만 다르다. 이렇게 문자열 안의 특정한 값을 바꿔야 할 경우가 있을 때 이것을 가능하게 해주는 것이 바로 문자열 포매팅 기법이다.

쉽게 말해 문자열 포매팅이란 문자열 안에 어떤 값을 삽입하는 방법이다. 다음 예를 직접 실행해 보면서 그 사용법을 알아보자.

문자열 포매팅 따라 하기

1. 숫자 바로 대입

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

위 예제의 결괏값을 보면 알겠지만 위 예제는 문자열 안에 정수 3을 삽입하는 방법을 보여 준다. 문자열 안에서 숫자를 넣고 싶은 자리에 %d 문자를 넣어 주고, 삽입할 숫자 3은 가장 뒤에 있는 % 문자 다음에 써 넣었다. 여기에서 %d는 문자열 포맷 코드라고 부른다.

문자열 포매팅

2. 문자열 바로 대입

문자열 안에 꼭 숫자만 넣으라는 법은 없다. 숫자 대신 문자열을 넣어 보자

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

위 예제에서는 문자열 안에 또 다른 문자열을 삽입하기 위해 앞에서 사용한 문자열 포맷 코드 %d가 아닌 %s를 썼다

※ 문자열을 대입할 때는 앞에서 배운 것처럼 큰따옴표나 작은따옴표를 반드시 써 주어야 한다.

3. 숫자 값을 나타내는 변수로 대입

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

4. 2개 이상의 값 넣기

그렇다면 문자열 안에 1개가 아닌 여러 개의 값을 넣고 싶을 땐 어떻게 해야 할까?

```
>>> 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.'.
```

문자열 포맷 코드

코드	설명
%s	문자열(String)
%c	문자 1개(character)
%d	정수(Integer)
%f	부동소수(floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 % 자체

문자열 포맷 코드의 적용

정수를 대입하는 경우 %d를 사용하고, 소수를 대입하는 경우 %f를 사용하면 됩니다.
문자열을 대입하는 경우에는 %s를 사용하면 됩니다.

```
year = 2016
month = 1
day = 16
day_of_week = "일"
# "오늘은 2016년 1월 16일 일요일입니다." 출력
print("오늘은 %d년 %d월 %d일 %s요일입니다." % (year, month, day, day_of_week))
# "오늘은 2016년 1월 17일 월요일입니다." 출력
print("오늘은 %d년 %d월 %d일 %s요일입니다." % (year, month, day + 1, "월"))
오늘은 2016년 1월 16일 일요일입니다.
오늘은 2016년 1월 17일 월요일입니다.
```

문자열 포맷 코드

[포매팅 연산자 %d와 %를 같이 쓸 때는 %%를 쓴다]

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

위 예문의 결괏값으로 당연히 "Error is 98%."가 출력될 것이라고 예상하겠지만 파이썬은 값이 올바르지 않다는 값 오류(Value Error) 메시지를 보여 준다.

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: incomplete format

이유는 문자열 포맷 코드인 %d와 %가 같은 문자열 안에 존재하는 경우, %를 나타내려면 반드시 %%로 써야 하는 법칙이 있기 때문이다. 이 점은 꼭 기억해 두어야 한다. 하지만 문자열 안에 %d 같은 포매팅 연산자가 없으면 %는 홀로 쓰여도 상관이 없다.

따라서 위 예를 제대로 실행하려면 다음과 같이 해야 한다.

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

'Error is 98%.'

포맷 코드와 숫자 함께 사용하기

1. 정렬과 공백

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

앞의 예문에서 %10s는 전체 길이가 10개인 문자열 공간에서 대입되는 값을 오른쪽으로 정렬하고 그 앞의 나머지는 공백으로 남겨 두라는 의미이다.

그렇다면 반대쪽인 왼쪽 정렬은 %-10s가 될 것이다. 확인해 보자.

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

hi를 왼쪽으로 정렬하고 나머지는 공백으로 채웠음을 볼 수 있다.

2. 소수점 표현하기

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

3.42134234를 소수점 네 번째 자리까지만 나타내고 싶은 경우에는 위와 같이 사용한다. 즉 여기서 '.'의 의미는 소수점 포인트를 말하고 그 뒤의 숫자 4는 소수점 뒤에 나올 숫자의 개수를 말한다. 다음 예를 살펴보자.

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

위 예는 숫자 3.42134234를 소수점 네 번째 자리까지만 표시하고 전체 길이가 10개인 문자열 공간에서 오른쪽으로 정렬하는 예를 보여 준다.

format 함수를 사용한 포매팅

문자열의 format 함수를 사용하면 좀 더 발전된 스타일로 문자열 포맷을 지정할 수 있다. 앞에서 살펴본 문자열 포매팅 예제를 format 함수를 사용해서 바꾸자

숫자 바로 대입하기

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

문자열 바로 대입하기

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

숫자 값을 가진 변수로 대입하기

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

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.'
```

2개 이상의 값을 넣을 경우 문자열의 {0}, {1}과 같은 인덱스 항목이 format 함수의 입력값으로 순서에 맞게 바뀐다. 즉 위 예에서 {0}은 format 함수의 첫 번째 입력값인 number로 바뀌고 {1}은 format 함수의 두 번째 입력값인 day로 바뀐다.

format 함수를 사용한 포매팅

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 사용하는 방법도 있다. {name} 형태를 사용할 경우 format 함수에는 반드시 name=value 와 같은 형태의 입력값이 있어야만 한다..

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.'

왼쪽 정렬

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

'hi '

:<10 표현식을 사용하면 치환되는 문자열을 왼쪽으로 정렬하고 문자열의 총 자릿수를 10으로 맞출 수 있다.

오른쪽 정렬

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

오른쪽 정렬은 :< 대신 :>을 사용하면 된다. 화살표 방향을 생각하면 어느 쪽으로 정렬되는지 바로 알 수 있을 것이다.

format 함수를 사용한 포매팅

가운데 정렬

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

:^ 기호를 사용하면 가운데 정렬도 가능하다.

공백 채우기

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

정렬할 때 공백 문자 대신에 지정한 문자 값으로 채워 넣는 것도 가능하다. 채워 넣을 문자 값은 정렬 문자 <, >, ^ 바로 앞에 넣어야 한다. 위 예에서 첫 번째 예제는 가운데(^)로 정렬하고 빈 공간을 = 문자로 채웠고, 두 번째 예제는 왼쪽(<)으로 정렬하고 빈 공간을 ! 문자로 채웠다.

소수점 표현하기

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

위 예는 format 함수를 사용해 소수점을 4자리까지만 표현하는 방법을 보여 준다. 앞에서 살펴보았던 표현식 0.4f를 그대로 사용한 것을 알 수 있다.

format 함수를 사용한 포매팅

>>> "{0:10.4f}".format(y) ' 3.4213'

위와 같이 자릿수를 10으로 맞출 수도 있다. 역시 앞에서 살펴본 10.4f의 표현식을 그대로 사용한 것을 알 수 있다.

{ 또는 } 문자 표현하기

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

{ }와 같은 중괄호(brace) 문자를 포매팅 문자가 아닌 문자 그대로 사용하고 싶은 경우에는 위 예의 {{ }}처럼 2개를 연속해서 사용하면 된다.

f 문자열 포매팅

파이썬 3.6 버전부터는 f 문자열 포매팅 기능을 사용할 수 있다. 파이썬 3.6 미만 버전에서는 사용할 수 없는 기능이므로 주의해야 한다.

다음과 같이 문자열 앞에 f 접두사를 붙이면 f 문자열 포매팅 기능을 사용할 수 있다.

>>> name = '홍길동'

>>> age = 30

>>> f'나의 이름은 {name}입니다. 나이는 {age}입니다.'

'나의 이름은 홍길동입니다. 나이는 30입니다.'

f 문자열 포매팅은 위와 같이 name, age와 같은 변수 값을 생성한 후에 그 값을 참조할 수 있다. 또한 f 문자열 포매팅은 표현식을 지원하기 때문에 다음과 같은 것도 가능하다. ※ 표현식이란 문자열 안에서 변수와 +, -와 같은 수식을 함께 사용하는 것을 말한다.

format 함수를 사용한 포매팅

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

딕셔너리는 f 문자열 포매팅에서 다음과 같이 사용할 수 있다.

※ 딕셔너리는 Key와 Value라는 것을 한 쌍으로 갖는 자료형이다. 02-5에서 자세히 알아본다.

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

정렬은 다음과 같이 할 수 있다.

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

format 함수를 사용한 포매팅

공백 채우기는 다음과 같이 할 수 있다.

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

소수점은 다음과 같이 표현할 수 있다.

```
>>> y = 3.42134234
>>> f'{y:0.4f}'  # 소수점 4자리까지만 표현
'3.4213'
>>> f'{y:10.4f}'  # 소수점 4자리까지 표현하고 총 자리수를 10으로 맞춤
'    3.4213'
```

f 문자열에서 { } 문자를 표시하려면 다음과 같이 두 개를 동시에 사용해야 한다.

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

지금까지는 문자열을 가지고 할 수 있는 기본적인 것에 대해 알아보았다. 이제부터는 문자열을 좀 더 자유자재로 다루기 위해 공부해야 할 것을 설명할 것이다

문자열 포맷 코드의 적용

소수형의 경우, 소수점 몇 번째 자리까지 출력할지 설정할 수 있습니다.

```
print(1.0 / 3)


print("1 나누기 3은 %f" % (1.0 / 3))
print("1 나누기 3은 %.4f" % (1.0 / 3))
print("1 나누기 3은 %.2f" % (1.0 / 3))
```

```
0.3333333333333333
1 나누기 3은 0.3333333333333333
1 나누기 3은 0.3333
1 나누기 3은 0.33
```

문자열 포맷팅

문자열 포맷팅의 개념을 이용하여, 아래의 문장들을 출력해보려고 합니다.

```
1시간에 5달러 벌었습니다.
5시간에 25달러 벌었습니다.
1시간에 5710.8원 벌었습니다.
5시간에 28554.0원 벌었습니다.
```

wage는 1시간에 얼마 버는지를 나타내는 값이고, exchange_rate은 1달러를 한국 돈으로 바꾸면 얼마인지 나타내는 값입니다. 이 경우, 1시간 동안 번 금액은 wage * 1의 결과값인 5달러이고, 이 금액을 한국 돈으로 환전을 하면 wage * 1 * exchange_rate의 결과값인 5710.8원.

아래 주어진 코드에 세 줄을 추가해서 문장들을 출력해보세요.

```
wage = 5                        # 시급 (1시간에 5달러)
exchange_rate = 1142.16         # 환율 (1달러에 1142.16원)

# "1시간에 5달러 벌었습니다." 출력
print("%d시간에 %d%s 벌었습니다." % (1, wage * 1, "달러"))
# "5시간에 25달러 벌었습니다." 출력
# 코드를 입력하세요.
# "1시간에 5710.8원 벌었습니다." 출력
# 코드를 입력하세요.
# "5시간에 28554.0원 벌었습니다." 출력
# 코드를 입력하세요.
```

1) 주어진 변수, 2) 문자열 포맷팅을 반드시 이용하셔야 합니다. 3) 콘솔 3-4번째 줄의 원화 금액의 경우, 소수점 첫 째자리까지만 출력되어야 합니다.

불 자료형이란?

불(bool) 자료형이란 참(True)과 거짓(False)을 나타내는 자료형이다. 불 자료형은 다음 2가지 값만을 가질 수 있다.

True - 참

False - 거짓

※ True나 False는 파이썬의 예약어로 true, false와 같이 사용하지 말고 첫 문자를 항상 대문자로 사용해야 한다.

다음과 같이 변수 a에는 True를, 변수 b에는 False를 지정해 보자.

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

따옴표로 감싸지 않은 문자열을 변수에 지정해서 오류가 발생할 것 같지만 잘 실행된다. type 함수를 변수 a와 b에 사용하면 두 변수의 자료형이 bool로 지정된 것을 확인할 수 있다.

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

※ type(x)는 x의 자료형을 확인하는 파이썬의 내장 함수이다.

불 자료형이란?

불 자료형은 조건문의 반환 값으로도 사용된다. 조건문에 대해서는 if문에서 자세히 배우겠지만 잠시 살펴보고 넘어가자.

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

1 == 1 은 "1과 1이 같은가?"를 묻는 조건문이다. 이런 조건문은 결과로 True 또는 False에 해당되는 불 자료형을 돌려준다. 1과 1은 같으므로 True를 돌려준다.

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

2는 1보다 크기 때문에 2 > 1 조건문은 True를 돌려준다.

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

2는 1보다 작지 않기 때문에 2 < 1 조건문은 False를 돌려준다

자료형의 참과 거짓

자료형에 참과 거짓이 있다? 조금 이상하게 들리겠지만 참과 거짓은 분명히 있다. 이는 매우 중요한 특징이며 실제로도 자주 쓰인다.

자료형의 참과 거짓을 구분하는 기준은 다음과 같다.

값	참 or 거짓
"python"	참
""	거짓
[1, 2, 3]	참
[]	거짓
()	거짓
{}	거짓
1	참
0	거짓
None	거짓

문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있으면(" ", [ ], ( ), { }) 거짓이 된다. 당연히 비어있지 않으면 참이 된다. 숫자에서는 그 값이 0일 때 거짓이 된다. 위 표를 보면 None이 있는데, 이것에 대해서는 뒷부분에서 배우니 아직은 신경 쓰지 말자. 그저 None은 거짓을 뜻한다는 것만 알아두자.

자료형의 참과 거짓

다음 예를 보고 참과 거짓이 프로그램에서 어떻게 쓰이는지 간단히 알아보자.

```
>>> a = [1, 2, 3, 4]
>>> while a:
...     print(a.pop())
4
3
2
1
```

먼저 a = [1, 2, 3, 4] 리스트를 하나 만들었다.

while문은 03장에서 자세히 다루겠지만 간단히 알아보면 다음과 같다. 조건문이 참인 동안 조건문 안에 있는 문장을 반복해서 수행한다.

```
while 조건문:
    수행할 문장
```

즉 위 예를 보면 a가 참인 경우에 a.pop( )을 계속 실행하라는 의미이다. a.pop( ) 함수는 리스트 a의 마지막 요소를 끄집어내는 함수이므로 리스트 안에 요소가 존재하는 한(a가 참인 동안) 마지막 요소를 계속해서 끄집어낼 것이다. 결국 더 이상 끄집어낼 것이 없으면 a가 빈 리스트([ ])가 되어 거짓이 된다. 따라서 while문에서 조건이 거짓이 되므로 중지된다. 위에서 본 예는 파이썬 프로그래밍에서 매우 자주 사용하는 기법 중 하나이다.

불린

연산자	뜻
>	초과 (greater than)
<	미만 (less than)
>=	이상 (greater than or equal to)
<=	이하 (less than or equal to)
==	같다 (equal to)
!=	같지 않다 (not equal to)

```
print(True)                    True
print(False)                   False
print()
print(2 > 1)    # 2는 1 초과이다      True
print(2 < 1)    # 2는 1 미만이다      False
print(3 >= 2)   # 3은 2 이상이다      True
print(3 <= 3)   # 3은 3 이하이다      True
print(2 == 2)   # 2는 2와 같다        True
print(2 != 2)   # 2는 2와 같지 않다   False
```

불 연산

자료형에 참과 거짓이 있음을 이미 알아보았다. bool 내장 함수를 사용하면 자료형의 참과 거짓을 식별할 수 있다.

다음 예제를 따라 해 보자.

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

'python' 문자열은 빈 문자열이 아니므로 bool 연산의 결과로 불 자료형인 True를 돌려준다.

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

'' 문자열은 빈 문자열이므로 bool 연산의 결과로 불 자료형인 False를 돌려준다.

위에서 알아본 몇 가지 예제를 더 수행해 보자.

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

불린

아래의 네 가지 규칙을 따르면 더 다양한 방법으로 참과 거짓을 나타낼 수 있습니다.
식이 and로 이어진 경우 왼쪽과 오른쪽 모두 True여야만 참이고, 나머지 False
식이 or로 이어진 경우 왼쪽과 오른쪽 모두 False여야만 거짓이고, 나머지 True
not True는 False고 not False는 True입니다.
괄호가 포함된 연산은, 괄호 안부터 값을 계산하고 그 후 괄호 밖과 연산합니다.

```
print(True and True)      True
print(True and False)     False
print(False and True)     False
print(False and False)    False
print()

print(True or True)     True
print(True or False)    True
print(False or True)    True
print(False or False)   False
print()

print(not True)    False
print(not False) True
```

불린

심화 예시를 살펴봅시다.

```
x = 3
print(x > 4 or (x < 2 or x != 3))

print((4 < 3 and 12 > 10) or 7 == 7)
print(not 4 < 3)
print(not not True)
```

```
False
True
True
True
```

type 함수

프로그래밍을 하다가 어떤 값이 무슨 자료형인지 기억나지 않을 때가 있습니다.
그럴 때에는 type라는 걸 쓰면 됩니다!

```
# 각 자료형 별 type 함수 써보기
print(type(1))        <class 'int'>
print(type(1.0))    <class 'float'>
print(type("1"))     <class 'str'>
print(type(2 > 4))  <class 'bool'>
print()

# 헷갈리는 두 자료형 비교
print(type("True"))   <class 'str'>
print(type(True))     <class 'bool'>
print()

# print 함수의 type 확인
print(type(print))   <class 'builtin_function_or_method'>
```

자료형 퀴즈

(1) 실행했을 때, True가 나오는 것을 모두 고르세요.

1. print(int(2.5) + int(3.8) > int(str(1) + str(2))
2. print((12 >= 10 and not 3 > 4) or 3 ** 2 != 9)
3. print(True and (True or False))
4. print(not True or (True and False))
5. print(False == False)

(해설)

(1) 부등호 왼쪽의 int(2.5) + int(3.8)은 2 + 3과 같기 때문에 5입니다. 부등호 오른쪽의 int(str(1) + str(2))는 int("1" + "2")와 같고, 이는 int("12")와 같기 때문에 12입니다. 5 > 12는 거짓이기 때문에 False가 나옵니다.

(2) 12 >= 10는 True이고, not 3 > 4 역시 not False이기 때문에 True입니다. 따라서 괄호 안의 값은 True and True로 True입니다. 3 ** 2은 9가 맞기 때문에, 3 ** 2 != 9라는 수식은 False입니다. True or False의 결과값으로 True가 나옵니다.

(3) 오른쪽 괄호 안의 True or False의 값은 True입니다. 괄호 밖과 연산하면 True and True이기 때문에 True가 나옵니다.

(4) 오른쪽 괄호 안의 True and False의 값은 False입니다. 괄호 밖과 연산하면 not True or False, 즉 False or False이기 때문에 False가 나옵니다.

(5) False가 False의 값과 같기 때문에 위 식은 True입니다.

따라서 (2), (3), (5)번이 정답입니다.

자료형 퀴즈

(2) 다음은 **type** 함수에 대한 내용입니다**.** 아래의 보기를 시행했을 때**,** 콘솔에 출력되는 내용과 잘못 짝지어진 것을 모두 고르세요**.**

print(type(4 / 2)) - <class 'int'>
print(type("True")) - <class 'bool'>
print(type(10 <= 7)) - <class 'bool'>
print(type(2.0 ** 3)) - <class 'float'>
print(type(2 * 3 == 6)) - <class 'int'>

(해설)
(1) 4 / 2는 2.0입니다. 따라서 <class 'int'>가 아니라 <class 'float'>가 출력됩니다.
(2) "True"는 문자열입니다. 따옴표가 없는 True가 불린이죠. 따라서 <class 'bool'>이 아니라 <class 'str'>가 출력됩니다.
(3) 10 <= 7은 결과값이 불린 값 False입니다. 옳은 보기입니다.
(4) 2 ** 3은 정수형 8이지만 2.0 ** 3은 소수형 8.0입니다. 옳은 보기입니다.
(5) 2 * 3 == 6은 결과값이 불린 값 True입니다. 따라서 <class 'int'>가 아니라 <class 'bool'>가 출력됩니다.
(1), (2), (5)번이 잘못되었습니다**.**

자료형 꿀팁

Floor Division
// 연산자는 나눗셈 연산 후 결과값을 반 내림 합니다. 즉 소수부분을 버리고 정수부분만 남겨둡니다. 이걸 **floor division**이라고 부릅니다.

```
# 아래 두 개의 결과는 같습니다.
x = int(5 / 2)  # 5를 2로 나눈 후 정수로 변환
print(x)    2
x = 5 // 2      # 5를 2로 나눈 후 소수 부분을 버림
print(x)    2
```

/와 //의 차이를 직접 확인해보세요.

```
print(7 / 4)        1.75
print(int(7 / 4))   1
print(7 // 4)        1
```

주의할 점은 소수형이 있을 경우 결과값이 소수형으로 나온다는 것입니다.

```
print(5.0 // 2)     2.0
print(5 // 2.0)     2.0
print(5.0 // 2.0)   2.0
```

반올림

round는 소수형을 반올림해줍니다.

```
print(round(1.421, 1))      # 1.421을 소수점 1째 자리로 반올림
print(round(2.7862, 2))     # 2.7562를 소수점 2째 자리로 반올림
print(round(3.141592, 4))   # 3.141592를 소수점 4째 자리로 반올림
print(round(4.34))          # 4.34를 소수점 0번째 자리로 반올림
1.4
2.79
3.1416
4
```

줄바꿈 기호 (Newline Character)

문자열 내에 \n 기호는 줄을 바꾸어주는 역할을 합니다.
키보드의 엔터키와 동일한 효과입니다.

```
print("안녕하세요\파이썬입니다\n여러분 모두를\n환영합니다")
안녕하세요
파이썬입니다
여러분 모두를
환영합니다
```

변수 정리

변수는 정보를 저장하고 쓸 수 있게 해주는 '이름표'같은 개념

변수는 왜 추상화의 한 종류일까요? 한 번 변수를 정의하고 나서 이후에 그것을 사용할 때, 변수의 이름만 알면 되기 때문

```
# 우사인 볼트의 평균 속도 = 10.438413361 m/s
# 1초에 달릴 수 있는 거리 (m/s)

speed = 10.438413361
print(speed * 60)   # 1분(60초)동안 간 거리
print(speed * 120)  # 2분(120초)동안 간 거리
print(speed * 180)  # 3분(180초)동안 간 거리
626.3048016600001
1252.6096033200001
1878.91440498
```

변수 정리

지정 연산자 (Assignment Operator)

```
x = 2 + 1
print(x)    3
x = x + 1
print(x)    4
```

수학에서 =은, "수학적으로 같은 값을 지닌다"는 뜻입니다.
이 기호는 지정 연산자(Assignment Operator)로서, 오른쪽에 있는 값을 왼쪽에 있는 변수에 '지정'해주는 역할

예를 들어 첫째 줄인 x = 2 + 1에서는 오른쪽에 있는 2 + 1 즉 3이, 왼쪽에 있는 변수인 x에 지정됩니다. 이에 둘째줄의 print(x)가 실행되었을 때, x가 저장하고 있는 3이 출력됩니다.

셋째 줄의 x = x + 1이라는 표현은 수학적으로 성립하지 않음
파이썬 에서는 x가 현재 3으로 지정되어 있기 때문에, x = x + 1의 오른쪽 값은 4가 됩니다.
4라는 값은 왼쪽 변수인 x에 지정됩니다.
따라서 네번째 줄의 print(x)가 실행되었을 때, x가 저장하고 있는 4가 출력됩니다.

변수 연습

name(이름), nationality(국적), phone_number(핸드폰 번호)라는 변수를 만들고, 여러분에게 알맞은 정보들을 지정하세요.
변수와 **문자열 포맷팅**을 이용하여 아래와 같이 출력되게 하세요.

```
Hi, my name is kildong. I'm from Korea.
My phone number is 010-1234-5678.
```

name을 문자열 “kildong "으로, nationality를 문자열 "Korea"로, 그리고 phone_number를 문자열 "010-1234-5678"로 지정

<주의> 문자열 덧셈 연산이 아닌, 문자열 포맷팅을 사용해야 합니다.
<주의> 문제의 조건에 정확히 따라주시기 바랍니다. 띄어쓰기도 일치

힌트
name = “kildong "
nationality = "Korea"
phone_number = "010-1234-5678“

힌트 2
문자열 포맷팅을 사용해야 합니다. 아래와 같은 형식으로 하면 됩니다.

```
print("Hi, my name is %s." % (name))
```