Chapter 02 - 자료형

- 자료와 자료형의 의미를 알아봅시다
- 문자열을 생성하는 방법과 문자열에 적용할 수 있는 연산자를 알아봅시다.
- 숫자를 생성하는 방법과 숫자에 적용할 수 있는 연산자를 알아봅시다.
- 변수를 선언하고 변수에 값을 할당하는 방법을 배웁니다.

Ch02-1 자료형과 문자열

- 프로그램이 처리할 수있는 모든 데이터에는 어떤 것들이 있으며
- **데이터**를 **보관** 및 **처리** 하는 간단한 방법에 대해 살펴봄.

시작하기 전에

컴퓨터에서 처리할 수 있는 데이터(자료, data)

- 문자, 사진, 숫자 등

데이터 처리

- 저장, (숫자) 계산 등

자료형과 기본 자료형

기능과 역활에 따라 자료를 구분(자료 형태에 따라 구분)

- 기본 자료형(data type)
  - 문자열(string) - 예, Hello, 안녕
  - 숫자(number) - 예, 12, 34.7
  - 불(boolean) - 예, True, False
- 기본 자료형을 조합하여 새로운 자료형을 만들 수 있음
  - 예, 김성필(73년생)
  - 예, 1973-00-00

자료(data)를 알아야 하는 이유

자료형(data type) 확인하기

파이썬에서 자료(data)의 형식(type)을 확인할 때 `type()` 함수 사용

```
1 print( type("안녕~") )
```

```
<class 'str'>
```

'str'이라고 출력되었는데 'string'의 약자를 출력한 것으로 "안녕~"이라는 자료(data)는 'string'형 데이터라는 것을 확인하였습니다. 'string'이라는 말은 컴퓨터 분야에서 문자열이라는 뜻입니다. 따라서 "안녕~"이라는 자료(data)는 문자열 형태의 데이터라는 뜻입니다.

[string]

```
print( type(73) )
```

```
<class 'int'>
```

'int'라고 출력되었는데 'integer'의 약자를 출력한 것입니다. 즉 정수형 데이터라는 뜻입니다.

[integer]

문자열(character string, 文字列) 만들기

```
"Hello", "안녕", "김 성필", "Good morning!"
```

위와 같이 문자열 형 데이터를 만들 때는 문자열을 이중 인용 부호로 감싸면 됩니다. 또는 단일 인용 부호로 감싸면 됩니다.

```
'Hello', '안녕', '김 성필', 'Good morning!'
```

[참고] - 나열(羅列)

```
print("안녕~")
print("좋은 아침이야")
```

```
안녕~
좋은 아침이야
```

큰 따옴표로 문자열 만들기

"안녕~"이라는 문자열 형 데이터와 "좋은 아침이야"라는 문자열 형 데이터를 생성해서(문자열을 이중 인용 부호로 감싸서) `print()` 함수에 전달해보겠습니다.

```
print("안녕~", "좋은 아침이야")
```

문자열 (형) 데이터 두 개를 "안녕~"과 "좋은 아침이야" `print()` 함수에 전달했으며 두 문자열 (형) 데이터를 구분하기 위해 쉼표(,)를 사용했습니다.

작은 따옴표로 문자열 만들기

작은 따옴표(')로도 문자열(형) 데이터를 만들 수 있습니다.

```
1 print('안녕~')
```

```
안녕~
```

문자열 내부에 따옴표 넣기

```
1 print("내 꿈은 '멋진 집'을 갖는거야!")
2 print('내 꿈은 "멋짓 집"을 갖는거라고!')
```

```
내 꿈은 '멋진 집'을 갖는거야!
내 꿈은 "먼짓 집"을 갖는거라고!
```

이스케이프 문자를 사용해 문자열 만들기

이스케이프 문자(escape character)는 백슬래시()기호와 함께 조합해서 사용하는 특수 문자. - 한글 키보드에서 백스래쉬(\)가 원화기호()로 표시됩니다.

```
1 print("내 꿈은 ₩"멋진 집₩"을 갖는거야!")
```

```
내 꿈은 "멋진 집"을 갖는거야!
```

```
1 print("내 꿈은 ₩'멋진 집₩'을 갖는거라고!")
```

```
내 꿈은 '멋진 집'을 갖는거라고!
```

다양한 이스케이프 문자가 있습니다.

- \n : 줄바꿈을 의미
- \t : 탭을 의미
- \\ : (백슬래시)를 의미

```
1 print("이름 나이 집업")
2 print("이름₩t나이₩t직업")
```

```
이름 나이 집업
이름	나이	직업
```

```
1 print("김성필")
2 print("50")
3 print("회사 대표")
```

```
김성필
50
회사 대표
```

```
print("김성필₩n")
print("50₩n")
print("회사 대표")
```

```
김성필

50

회사 대표
```

print() 함수는 소괄호 안에 있는 내용을 출력 후 **그 다음 출력 위치를 다음 줄 맨 처음으로 바꾼다(줄바꿈 발생)**. 그런데 print() 함수에 "김성필\n"의 문자열 데이터를 전달하게 되면 "김성필"이라는 문자열을 화면에 출력하고 이스케이프 문자 "\n"은 줄바꿈으로 해석해서 다음 출력 위치를 다음 줄 맨 처음으로 바꾼다. 이렇게 "김성필\n"을 출력 후에 print() 함수는 '그 다음 출력 위치를 다음 줄 맨 처음으로 바꿈' 동작을 하게 된다.

```
print("김성필₩n50₩n회사대표")
```

```
김성필
50
회사대표
```

▼ ◂참고▸

print() 함수는 소괄호 안에 표현된 데이터를 화면에 출력하고 마지막 동작으로 향후 발생할 다음 출력을 위치를 다음 줄로 이동 후 종료하는 것이 기본 설정입니다.

그런데 이 마지막 동작, "향후 발생할 다음 출력 위치를 다음 줄로 이동"을 다른 동작으로 바꿀 수 있습니다. 아래 코드 두 줄에 보인 print() 함수는 기본 동작을 수행하고 있습니다.

```
print("인공지능 :")
print("머신러닝", "딥러닝")
```

```
인공지능 :
머신러닝 딥러닝
```

print() 함수의 마직막 동작은 변경하는 코드를 아래 보이고 있습니다.

```
print("인공지능 :", end=" ")
print("머신러닝", "딥러닝")
```

```
인공지능 : 머신러닝 딥러닝
```

print() 함수는 소괄호 안에 표현된 데이터를 화면에 출력하고 마지막 동작(end)에 대한 특별한 언급이 없었다면 "다음 출력 위치를 다음 줄로 이동" 동작을 했을텐데, 마지막 동작(end=" ")가 지정되었기 때문에 지정된 동작을 수행합니다. end=" " 은 공백(스페이스)를 출력하라는 뜻입니다. 따

라서 기본 설정된 마지막 동작, "다음 출력 위치를 다음 줄로 이동" 동작을 하지 않고 공백만 하나 출력하고 print() 함수의 동작외 모두 완료 됩니다.

```
print("인공지능 :", end="₩t")
print("머신러닝", "딥러닝")
```

```
인공지능 :	머신러닝 딥러닝
```

여러 줄 문자열 만들기

```
print("한 번 더 나에게₩n질풍 같은 용기를₩n거친 파도에도 굴하지 않게₩n드넓은 대지에 다시 새길 희망
```

```
한 번 더 나에게
질풍 같은 용기를
거친 파도에도 굴하지 않게
드넓은 대지에 다시 새길 희망을

-질풍가도-
```

여러 줄의 문자열은 큰따옴표(또는 작은따옴표)를 세번 반복하여 문자열 데이터를 생성할 수 있습니다.

```
print("""한 번 더 나에게
질풍 같은 용기를
거친 파도에도 굴하지 않게
드넓은 대지에 다시 새길 희망을

-질풍가도-""")
```

```
한 번 더 나에게
질풍 같은 용기를
거친 파도에도 굴하지 않게
드넓은 대지에 다시 새길 희망을

-질풍가도-
```

문자열 연산자

자료(데이터)의 자료형에 따라 적용할 수 있는 연산자가 정해져 있습니다.

숫자형 데이터 2와 숫자형 데이터3은 더할 수 있습니다.

```
2 + 3
```

```
2 / 3
```

그런데 문자열 데이터 "김"과 문자열 데이터 "성필"에 대해 아래 연산은 python에서 가능하지도 않고 의미도 알수 없습니다.

```
"김" / "성필"
```

숫자형 데이터에 적용할 수 있을 것으로 짐작되는 연산(자)은 짐작이 됩니다.예를 들어 사칙연산이 가능할 것입니다. 그렇다면 문자열 데이터에 적용할 수 있는 연산은 어떤 것이 있으며 그러한 연산을 어떤 기호로 표시하는지 살펴 보겠습니다.

문자열 연결 연산자 : +

```
"문자열" + "문자열"
```

```
1 print( "김" + "성필" )
```

```
김성필
```

```
1 print( "제 꿈은" + " 자유입니다!" )
```

```
제 꿈은 자유입니다!
```

위 두 문장(statement), 위 두 줄의 코드의 출력 결과 "김성필"과 "제 꿈은 자유입니다!"에서 뛰어 쓰기와 코드의 관계를 살펴 보십시요. " 자유입니다!"라는 문자열 데이터에는 공백(스페이스)이 포함되어 있는데 이 공백(스페이스)를 지워 "자유입니다!"로 데이터를 수정한 후 출력 결과를 살펴 보시기 바랍니다.

```
1 print( "제 꿈은" + "자유입니다!" )
```

```
제 꿈은자유입니다!
```

그리고 아래 코드의 출력 결과와 `print( "김" + "성필" )`의 문자(statement)의 출력 결과와 비교해 보십시요.

```
1 print( "김", "성필" )
```

```
김 성필
```

아래 코드의 실행 결과를 예측해 보십시요.

```
print("오늘 최고 기온은 " + 25 + "도 입니다")
```

```
1 print("오늘 최고 기온은 " + 25 + "도 입니다")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-28-b36f6d41a62e> in <module>
----> 1 print("오늘 최고 기온은 " + 25 + "도 입니다")

TypeError: can only concatenate str (not "int") to str
```

TypeError: can only concatenate str (not "int") to str

위와 같은 에러(타입 에러)가 발생했습니다. 타입 에러(TypeError)는 통상 데이터 타입이 해당 연산에 적합하지 않기 때문에 발생합니다. 이런(류)의 에러는 python으로 프로그래밍하면서 자주 발생할 수 있으니 그 의미를 명확히 이해하고 넘어가시길 바라겠습니다.

`TypeError: can only concatenate str (not "int") to str` 의 표현을 해석해 보면 can 할수 있답니다. only 단지, cancatenate 붙일 수, str(ing) to str(ing) 문자열 대 문자열 ㅎㅎㅎ

데이터 형 에러 : "문자열 대 문자열만 붙일 수 있어요." 다시 표현하면 데이터 형 에러 : "문자열과 문자열만 붙일 수 있어요. 전수와 문자열을 붙일 수 없어요"라고 하는 것입니다.

"문자열과 문자열만 이을 수 있어요"

[concatenate]

문자열 반복 연산자 : *

```
1 print("안녕"*3)
2 print("안녕 "*3)
```

```
안녕안녕안녕
안녕 안녕 안녕
```

숫자와 문자열의 위치가 '숫자*문자열' 형태이어도 동일한 의미입니다.

```
1 print(3*"안녕 ")
```

```
안녕 안녕 안녕
```

문자열 선택 연산자(인덱싱):[ ]

홍콩 여배우, "구숙정"이라는 사람이 있습니다. 참고로 68년생입니다. 누나? 언니? 이모?

문자열 데이터 "구숙정"은 세 개의 글자로 구성되어 있는데 첫[0] 번째 글자는 "구", 두[1] 번째 글자는 "숙", 세[2] 번째 글자는 "정"입니다.

데이터	'구'	'숙'	'정'
인덱스	0	1	2

여기서 각각의 글자(요소)에 접근하기 위한 숫자 0, 1, 2를 인텍스(index)라고 합니다.

```
1 print( "구숙정"[0] )
```

```
구
```

```
1 print( "구숙정"[1] )
```

```
숙
```

```
1 print( "구숙정"[2] )
```

```
정
```

데이터	'구'	'숙'	'정'
인덱스 부여 방식 A	0	1	2
인덱스 부여 방식 B	-3	-2	-1

짧은 테이프에 "구숙정"이라고 쓰고 테이프의 앞과 뒤를 붙여 원으로 만들었다고 가정합니다. 문자열 "구숙정"에서 "구" 자리를 인덱스 0 위치라고 하면 "구"왼쪽에는 "정"이라는 글자가 위치하며 그 자리는 0의 왼쪽 자리 즉 -1의 위치로 볼 수 있습니다. 위 표에 보인 것과 같이 데이터의 위치를 표현하는 방법이 두 가지 있으며 상황에 따라 적합한 방법을 사용하면 됩니다. '인덱스 부여 방식 B'라고 명명한 방식으로 인덱싱하는 것이 '인덱스 부여 방식 A'보다 더 적절할 때가 있습니다.

```
1 print("구숙정"[0])
2 print("구숙정"[1])
3 print("구숙정"[2])
4 print()
5 print("구숙정"[-3])
6 print("구숙정"[-2])
7 print("구숙정"[-1])
```

```
구
숙
정

구
숙
정
```

"소호강호"라는 문자열 데이터를 각각의 인덱싱 방법으로 표현했을 때 인덱스의 값은 아래 표와 같습니다.

데이터	'소'	'호'	'강'	'호'
인덱스 부여 방식 A	0	1	2	3
인덱스 부여 방식 B	-4	-3	-2	-1

인덱스 0에 해당하는 글자는 "소", 인덱스 1에 해당하는 글자는 "호", ..., 인덱스 3에 해당하는 글자는 "호"가 됩니다. "소"라는 데이터의 위치(인덱스)는 0 또는 -4로 표현할 수 있습니다.

```
1 print("소호강호"[0])
2 print("소호강호"[1])
3 print("소호강호"[2])
4 print("소호강호"[3])
5 print()
6 print("소호강호"[-4])
7 print("소호강호"[-3])
8 print("소호강호"[-2])
9 print("소호강호"[-1])
```

```
소
호
강
호

소
호
강
호
```

- 참고 -

[질풍가도] 쾌걸 근육맨 2세는 2000년대 투니버스에서 방영된 애니

[다른 가수가 부른 질풍가도]

문자열 범위 선택 연산자(스라이싱): [:]

"질풍가도(쾌걸 근육맨)"이라는 문자열 데이터를 가지고 문자열 내에서 범위를 지정하여 데이터를 추출하는 방법을 알아보겠습니다. 예를 들어 "질풍가도(쾌걸 근육맨)"이라는 문자열 데이터에서 "질풍가도"만 추출하거나 "(쾌걸 근육맨)"만 분리해 보는 것입니다. 다른 예를 들자면 "김성필 1973-00-00"에서 생년월일 데이터만 분리 추출을 어떻게 할 것인가에 대해 알아보겠습니다.

```
print("질풍가도(쾌걸 근육맨)"[0])   # 인덱스 0은 맨 처음
print("질풍가도(쾌걸 근육맨)"[-1])  # 인덱스 -1은 맨 마지막
```

```
질
)
```

```
print("질풍가도(쾌걸 근육맨)"[3])
print("질풍가도(쾌걸 근육맨)"[5])
```

```
도
쾌
```

문자열 데이터, "질풍가도(쾌걸 근육맨)"에서 "질풍가도"만 추출하겠습니다. 다시 말해서 인덱스 0부터 인덱스 3까지의 데이터만 추출하면 되겠습니다.

```
"질풍가도(쾌걸 근육맨)"[0~3]
```

위 코드에서 `[0~3]` 표현은 파이썬 문법에 맞지 않습니다. 인덱스 0부터 인덱스 3까지의 의미를 담고 있는 표현은 `[0:4]` 입니다. `[0:3]` 이 아닙니다.

```
[start:end]
```

위 표현에서 `start` 는 지정하려는 범위의 시작 인덱스이고 `end` 는 지정하려는 범위의 끝 인덱스보다 1 더 큰 수입니다. 지정하려는 범위가 인덱스 0부터 인덱스 3까지라면 `start` 의 값은 0이고 `end` 의 값은 지정하려는 범위의 끝 인덱스 보다 1 더 큰 4가 되어야 합니다.

```
"질풍가도(쾌걸 근육맨)"[0:4]   # 0:4 --> 0, 1, 2, 3
```

```
'질풍가도'
```

이번에는 문자열 데이터 "질풍가도(쾌걸 근육맨)"에서 "(쾌걸 근육맨)"만 분리 추출해 보겠습니다. 먼저 추출할 데이터의 인덱스 범위를 생각해 보겠습니다. 문자열 "질풍가도(쾌걸 근육맨)"의 인덱스 4부터 인덱스 11까지 분리 추출하면 원하는 결과를 얻을 수 있겠습니다. 그러면 `start` 의 값은 지정하려는 범위의 시작 인덱스인 4가 되어야 하고 `end` 는 지정하려는 범위의 끝 인덱스 11 보다 1 더 큰 수인 12가 되면 "질풍가도(쾌걸 근육맨)"에서 "(쾌걸 근육맨)"을 분리 추출할 수 있습니다.

```
print( "질풍가도(쾌걸 근육맨)"[4:12] )
```

```
(쾌걸 근육맨)
```

다시 "질풍가도(쾌걸 근육맨)"에서 "질풍가도"만 추출하는 코드를 살펴보겠습니다.

```
print( "질풍가도(쾌걸 근육맨)"[0:4] )
print( "질풍가도(쾌걸 근육맨)"[:4] )
```

```
질풍가도
질풍가도
```

위 코드 셀에 있는 두 명령은 동일한 문자열을 출력했습니다. 즉 `[0:4]`와 `[:4]`가 동일한 의미라는 것을 짐작할 수 있습니다. `[start:end]` 형태에서 start의 값이 명시적으로 표현되지 않으면 `start` 값은 0이 됩니다. 좀 더 정확히 이야기 하면 `start` 값이 명확하게 언급되지 않은면 'start'에는 맨 처음 데이터(글자)의 인덱스가 할당됩니다. '같은 의미의 말 아닌가?' 생각되실 수 있는데 여기서는 이쯤 설명하고 다음으로 가겠습니다.

또 다시 "질풍가도(쾌걸 근육맨)"에서 "(쾌걸 근육맨)"만 추출하는 코드를 살펴보겠습니다.

```
1 print( "질풍가도(쾌걸 근육맨)"[4:12] )
2 print( "질풍가도(쾌걸 근육맨)"[4:] )
```

```
(쾌걸 근육맨)
(쾌걸 근육맨)
```

위 코드 셀에 있는 두 명령 실행 결과는 동일합니다. `[4:12]`는 인덱스 시작 범위를 0부터 12까지 설정한 것입니다. "질풍가도(쾌걸 근육맨)"에서 인덱스 12는 맨 마지막 글자의 인덱스입니다. 따라서 `[4:]` 표현처럼 `end`의 값이 생략되면 `end` 값은 문자열의 맨 마지막 글자에 대한 인덱스가 됩니다.

Quiz

문자열 데이터, "질풍가도(쾌걸 근육맨)"에서 "쾌걸 근육맨"만 분리하여 출력(print)하는 코드를 작성하세요.

```
1 print("질풍가도(쾌걸 근육맨)"[5:11])
```

```
쾌걸 근육맨
```

아래 코드 실행 결과는 어떻게 될까요?

```
1 print( "한번 더 나에게 질풍같은 용기를"[5:] )
```

```
나에게 질풍같은 용기를
```

IndexError(index out of range) 예외 - 예외 처리

아래 코드가 실행되었을 때, 어떤 결과가 출력될 것인지 예상해 보세요.

```
print("한번 더 나에게 질풍같은 용기를"[9:17])
print("한번 더 나에게 질풍같은 용기를"[18])
```

```
1 print("한번 더 나에게 질풍같은 용기를"[9:17])
```

```
질풍같은 용기를
```

```
1 print("한번 더 나에게 질풍같은 용기를"[18])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-62-8c337755c40e> in <module>
----> 1 print("한번 더 나에게 질풍같은 용기를"[18])

IndexError: string index out of range

 SEARCH STACK OVERFLOW
```

인덱스 에러(IndexError)가 발생했습니다.

"IndexError: string index out of range"

해석해보면 문자열 인덱스 out of range, 문자열 인덱스 범위 밖, "문자열 인덱스가 범위 밖이다"

"한번 더 나에게 질풍같은 용기를"은 공백(스페이스)를 포함하여 총 18개 글자로 구성되어 있으며 인덱스 0은 "한"에 해당하고 인덱스 1은 "번", ..., 인덱스 17은 맨 마지막 글자 "를"에 해당한다. 그러니까 위 문자열 데이터의 마지막 인덱스는 17인데 있지도 않은 인덱스 18에 해당하는 데이터를 추출하려고 하니 "(문자열)인덱스가 범위(0~17)를 벗어났다"라고 에러를 출력한 것입니다.

"IndexError: string index out of range"이 에러도 python으로 프로그래밍 할 때 종종 만나게 되는 에러이니 확실하게 이해하고 다음으로 진행하시길 바랍니다.

Quiz

다음 코드가 실행되었을 때 어떤 결과가 발생하는가?

```
print("거친 파도에도" + "굴하지 않게"[:7])
print("거친 파도에도" + " 굴하지 않게"[:7])
print("거친 파도에도 굴하지 않게"[0:])
print("거친 파도에도 굴하지 않게"[3:])
```

```
1 print("거친 파도에도" + "굴하지 않게"[:7])
2 print("거친 파도에도" + " 굴하지 않게"[:7])
3 print("거친 파도에도 굴하지 않게"[0:])
4 print("거친 파도에도 굴하지 않게"[3:])
```

```
거친 파도에도굴하지 않게
거친 파도에도 굴하지 않게
거친 파도에도 굴하지 않게
파도에도 굴하지 않게
```

문자열의 길이 구하기

`len()` **함수**로 문자열 데이터의 길이(문자 개수)를 구할 수 있음.

함수는 수학 시간에 처음 들었던 말입니다. 수학 선생님께서 "함수를 영어로 바꾸면 뭐지?" 우리는 대답합니다. "function요"

영어 수업 시간입니다. 영어 선생님이 "function이 한국말로 뭐지?" 우리는 대답합니다. **기능**요.

프로그래밍 수업 시간입니다. "funtion이 뭐지?" 뭐라고 대답하고 싶습니까?

이후 function(함수)라는 말이 자주 등장하는데, 한 마디로 기능 블록입니다. 어떤 일을 하도록 작성된 기능 블록입니다.

`len()`는 문자열 데이터의 길이를 측정할 수 있는 기능을 갖은 코드 블록, 함수(function)입니다.

```
1 print( len("질풍가도") )
```

```
4
```

Ch02-2 숫자

숫자

- 소수점이 없는 숫자
- 소수점이 있는 숫자

시작하기 전에

숫자

- 소수점이 없는 숫자:정수(integer)형
- 소수점이 있는 숫자:실수(floating point)형

정수에 대해서는 굳이 설명하지 않아도 충분히 이해하시겠습니다만 예를 들자면 -1, 0, 1, 2, 128, -61과 같은 숫자입니다. ^^

52.273을 0.52273×10^2와 같이 소수점의 위치를 바꿔 표현해도 같은 숫자를 나타내다고 해서 floating point(부동 소수점)라고 부른다 정도로만 이해하고 넘어가겠습니다.

숫자의 종류

앞서 살펴본 것과 같이 크게 소수점이 없는 숫자와 소수점이 있는 숫자에 대해 사펴 보겠습니다.

```
1 print(64)
2 print( type(64) )
```

```
64
<class 'int'>
```

```
1 print(3.14)
2 print( type(3.14) )
```

```
3.14
<class 'float'>
```

▾ ◂참고▸

```
print(5.1)      # 5.1
print(5.1e2)    # 510.0
print(5.1e-2)    # 0.051
```

5.1e2은 5.1×10^2라는 의미 따라서 510.0입니다. 5.1e-2는 5.1×10^{-2}라는 의미 따라서 0.051입니다.

```
1 print(5.1)
2 print(5.1e2)
3 print(5.1e-2)
```

```
5.1
510.0
0.051
```

```
1 print('원주율 =', 3.14156)
```

```
원주율 = 3.14156
```

```
1 print('1[s] =', 1000, '[ms]')
```

```
1[s] = 1000 [ms]
```

▾ 숫자 연산자

▾ 사칙 연산자 : +,-,*,/

```
1 print(3+2)
2 print(3-2)
3 print(3*2)
4 print(3/2)
```

```
5
1
6
1.5
```

```
1 print("3+2=", 3+2)
2 print("3-2=", 3-2)
3 print("3*2=", 3*2)
4 print("3/2=", 3/2)
```

```
3+2= 5
3-2= 1
3*2= 6
3/2= 1.5
```

정수 나누기 연산자 : //

```
1 print("3 / 2 =", 3/2)
2 print("3 // 2 =", 3//2) # 몫
```

```
3 / 2 = 1.5
3 // 2 = 1
```

나머지 연산자: %

```
1 # 2의 배수를 2로 나눈었을 때 나머지는?
2 print("4 % 2 =", 4 % 2)
3 print("2 % 2 =", 2 % 2)
4 print("0 % 2 =", 4 % 2)
5 print("-2 % 2 =", -2 % 2)
```

```
4 % 2 = 0
2 % 2 = 0
0 % 2 = 0
-2 % 2 = 0
```

```
1 # 작수 또는 홀수를 2로 나눈 후 나머지는?
2 print("0 % 2 =", 0 % 2)
3 print("1 % 2 =", 1 % 2)
4 print("2 % 2 =", 2 % 2)
5 print("3 % 2 =", 3 % 2)
6 print("4 % 2 =", 4 % 2)
7 print("5 % 2 =", 5 % 2)
8 print("6 % 2 =", 6 % 2)
9 print("7 % 2 =", 7 % 2)
10 print("8 % 2 =", 8 % 2)
11 print("9 % 2 =", 9 % 2)
```

```
0 % 2 = 0
1 % 2 = 1
2 % 2 = 0
3 % 2 = 1
4 % 2 = 0
5 % 2 = 1
6 % 2 = 0
7 % 2 = 1
```

```
8 % 2 = 0
9 % 2 = 1
```

제곱 연산자: **

2^4을 python 코드로 표현할 때

```
2**4
```

로 표현함.
$2^4 = 16$임

```
1 print(2**2)
2 print(2**4)
```

```
4
16
```

연산자의 우선순위

```
5 + 3 * 2
```

위 식의 계산(연산) 결과는 얼마일까요? 네~ 11 맞습니다. 사칙(+, -, *, /)연산에서 곱하기나 나누기를 먼저 하고 나서 덧셈이나 뺄셈을 합니다.

```
(5 + 3) * 2
```

위 식의 연산 결과는 얼마일까요? 네~ 16 맞습니다. (소)괄호 안의 연산부터 실생합니다.

◂참고 - 문자열 연산자의 우선 순위▸

```
1 print("안녕" + "하세요" * 3)
```

```
안녕하세요하세요하세요
```

문자열 연산에서 * 연산이 + 연산보다 우선 처리된다는 것을 확인했습니다. 그런데 나중에 연산자 우선 순위가 잘 기억나지 않는다면 아래와 같이 소괄호를 활용하여 코딩하면 되겠습니다.

```
1 print( ("안녕"+"하세요") * 3)
2 print( "안녕" + ("하세요" * 3) )
```

```
안녕하세요안녕하세요안녕하세요
안녕하세요하세요하세요
```

TypeError 예외

서로 다른 자료를 연산하면 TpyeError라는 예외가 발생합니다.

```
1 s = '문자열'
2 i = 12
3 s + i
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1-9a29bb402c06> in <module>
      1 s = '문자열'
      2 i = 12
----> 3 s + i

TypeError: can only concatenate str (not "int") to str
```

SEARCH STACK OVERFLOW

Ch02-3 변수와 입력

시작하기 전에

변수는 데이터를 보관하는 하는 기능이 있음.

```
a = 3.14
```

위 코드는 변수 a 에 숫자 데이터 3.14 를 할당하라는 의미입니다. 다시 말해서 변수 a 라는 녀석에게 숫자 데이터 3.14 를 기억하고 있으라고 명령하는 것입니다.

참고 - 여기서 변수라고 부르고 있는 녀석은, 파이썬에서 이 녀석은 데이터를 기억하는 능력만 있는 것이 아닙니다. 향후 이 녀석에 대한 설명을 조금 더 자세히 하겠습니만 지금은 데이터를 보관하는 장소라고 생각하시면 되겠습니다.

변수 만들기/사용하기

원주율(π)를 저장하기 위해 pi 라는 이름의 변수에 숫자 데이터 3.14 를 저장하는 코드를 만들어 보겠습니다.

```
pi = 3.14
```

변수 pi 라는 녀석은 숫자 데이터 3.14 를 가지고 있게 됩니다.

```
1 pi = 3.14     # 변수 선언 : "이름이 pi라는 녀석에게 숫자 데이터 3.14를 가지고 있으라고 한다"
2 print(3.14)
```

```
print( pi )   # 변수 참조 : 변수, pi라는 녀석이 가지고 있는 값을 출력(print)해라,
              #          : 변수, pi라는 녀석이 가지고 있는 값을 쓰겠다.
```

```
3.14
3.14
```

어떤 원의 반지름의 길이가 10이라고 가정하고 반지름 값을 저장하기 위해 변수 r 을 생성하고 r 에 10을 할당하는 코드를 작성하겠습니다. 그리고 원의 둘레와 원의 면적(넓이)를 계산해 보겠습니다.

원의 둘레는 $2\pi r$이고 원의 면적은 πr^2입니다.

```
r = 10

#변수 참조 - 변수가 가지고 있는 값을 사용
print('원주율 =', pi)
print('반지름 =', r)
print('원의 둘레 =', 2*pi*r)
print('원의 면적 =', pi*r*r)
```

```
원주율 = 3.14
반지름 = 10
원의 둘레 = 62.800000000000004
원의 면적 = 314.0
```

변수는 내부에 들어 있는 데이터의 형에 따라 사용할 수 있는 연산자가 다릅니다. 숫자형 데이터를 가지고 있는 변수 녀석은 숫자형 데이터에 적용할 수 있는 연산자를 사용할 수 있습니다. 아래 코드를 확인하세요.

```
print(r + 1.0)
```

```
11.0
```

변수 r 은 숫자형 데이터를 가지고 있습니다. 따라서 덧셈(+) 연산자를 사용하여 변수 r 이 가지고 있는 값과 숫자 데이터 1을 더한 연산이 가능합니다. 또 뺄셈(-)도 가능할 것입니다.

```
print(r-1.0)
```

```
9.0
```

그렇다면 아래 코드의 실행 결과는 어떻게 될까요?

```
s = "질풍가도"
print( s + "(괘걸 근육맨)" )
```

```
1 s = "질풍가도"
2 print( s + "(괘걸 근육맨)" )
```

```
질풍가도(괘걸 근육맨)
```

복합 대입 연산자

```
a += 10
```

위 코드의 의미는 `a = a + 10`의 의미입니다. 그렇다면 아래 코드가 실행되었을 때 출력되는 값은 얼마이겠습니까?

```
a = 10
a += 10
print(a)
```

여러분의 예상과 출력 결과 값이 같았으면 좋겠습니다.

```
1 a = 10
2 a += 10   # a = a + 10
3 print(a)
```

```
20
```

위와 같은 복합연산자(+=)외에 도 여러 가지 복합연산자가 있습니다.

연산자 이름	설명
+=	숫자 덧셈 후 대입
-=	숫자 뺄셈 후 대입
*=	숫자 곱셈 후 대입
/=	숫자 나눗셈 후 대입
%=	숫자의 나머지를 구한 후 대입
**=	숫자 제곱 후 대입

```
1 a = 0
2 a += 10     # a = a + 10
3 print( a )
4
5 a -= 4      # a = a - 4
6 print( a )
7
8 a *= 2      # a = a * 2
9 print( a )
10
11 a /= 2     # a = a / 2
12 print( a )
13
```

```
a %= 4      # a = a % 4
print( a )

a **= 4     # a = a ** 4
print( a)
```

```
10
6
12
6.0
2.0
16.0
```

위에서는 숫자 데이터에 적용할 수 있는 복합 대입 연산자에 대해 살펴 봤습니다. 문자열에 대해서도 적용할 수 있는 복합 대입 연사자가 있습니다.

연산자 이름	설명
+=	문자열 연결 후 대입
*=	문자열 반복 후 대입

```
string = "아자 아자"
string += "!"         # string = string + "!"
string += "!"         # string = string + "!"

print(string)
print("OK!")
```

```
아자 아자!!
OK!
```

```
string = "아자 아자"
string += "₩n"
string += "₩n"

print(string)
print("OK!")
```

```
아자 아자


OK!
```

사용자 입력:input()

실무에서는 프로그램을 만들 때 사용자로부터 명령 프롬프트에서 어떤 값을 입력 받는 일이 거의 없습니다만 프로그램 작성을 학습하는 과정에서는 사용자로부터 데이터를 입력 받는 경우가 종종 있습니다. 이 때 편리하게 사용할 수 있는 함수가 `input()` 입니다.

input() 함수로 사용자 입력 받기

```
1 input("1 이상 10 이하의 정수를 입력하세요 : ")
```

```
1 이상 10 이하의 정수를 입력하세요 : 12
'12'
```

print() **함수**에 이어 두 번째로 만난 **함수**는 input() 입니다. 함수는 기능입니다. 함수 이름(input)만 봐도 input() 함수의 기능이 짐작이 됩니다. 사용자로부터 입력 데이터를 받는 기능을 가지고 있는 것을 쉽게 짐작할 수 있습니다.

함수(function)들 중 일부는 주어진 기능을 수행한 후 어떤 값을 반환하기도 하기도 합니다.
input() 함수는 "사용자로부터 데이터를 입력 받고 그 입력 받은 값은 반환합니다."

```
1 value = input("1 이상 10 이하의 정수를 입력하세요 : ")
2 print(value)
```

```
1 이상 10 이하의 정수를 입력하세요 : 1
1
```

input() 함수의 입력 자료형

앞서 input() 함수의 결과를 string

```
1
```

문자열을 숫자로 바꾸기

input() 함수의 입력 자료형은 항상 문자열이기 때문에 입력받은 문자열을 숫자로 변환해야 연산에 활용할 수 있습니다. 문자열 형 데이터를 숫자 형 데이터로 변환하는 것과 같이 형 변환을 하는 것을 캐스트cast라고 합니다.

- int() : 문자열(형) 데이터를 int(형) 데이터로 변환
- float(): 문자열(형) 데이터를 float(형) 데이터로 변환

```
1 # int() 함수 활용하기
2 2
3
4 string_a = input('입력> ')
5 int_a = int(string_a)
6
7 string_b = input('입력> ')
8 int_b = int(string_b)
9
10 print(string_a + string_b)
11 print(int_a + int_b)
```

```
입력> 1
입력> 2
```

```
12
3
```

```
# int() 함수와 float() 함수 활용하기

output_a = int('52')
output_b = float('3.1415')

print(type(output_a), output_a)
print(type(output_b), output_b)
```

```
<class 'int'> 52
<class 'float'> 3.1415
```

Quiz

숫자 두 개를 받아 사칙 연산 결과를 출력하는 코드를 작성하시요.

ValueError 예외

데이터의 자료 형을 변환하려고 할 때 변환 분가능한 변환을 시도할 때 ValueError 예외가 발생합니다. 이러한 예외는 다음 두 가지로 구분할 수 있습니다.

첫째, 숫자가 아닌 것을 숫자로 변환하려고할 때,

```
int('Good job!')
float('내 사랑 내 곁에 - 김현식')
```

둘째, 소수점이 있는 숫자 형태의 문자열을 int()함수로 정수로 변환하려고 할 때

```
int('3.14')
int('43.89')
```

변수 x에 정수가 들어 있는지 소수점을 포함한 수가 들어 있는지 알 수 없을 때는 어떻게 하죠?

```
float(x)
```

실수형으로 변환을... ^^

```
1
```

숫자를 문자열로 바꾸기

문자열 형(타입)의 데이터를 숫자 형(타입) 데이터로 바꾸는 방법에 대해 위에서 살펴봤습니다. 이번에는 수자를 문자열 형(타입) 데이터로 바꾸는 방법에 대해 살펴 보겠습니다.

```
output = str(32)
print(type(output))
print(output)
```

```
<class 'str'>
32
```

```
output = str(3.1415)
print(type(output))
print(output)
```

```
<class 'str'>
3.1415
```

inch 단위를 cm 단위로 변경하기

1[inch]는 2.54[cm]입니다. 인치 단위의 수를 입력 받아서 cm 단위로 환산 출력하는 코드를 작성해 보죠.

```
inch = input('inch 단위의 숫자를 입력하세요. cm 단위로 변환하겠습니다 : ')
n_inch = float(inch)
n_cm = n_inch * 2.54
print(n_cm, '[cm]입니다.')
```

```
inch 단위의 숫자를 입력하세요. cm 단위로 변환하겠습니다 : 28.5
72.39 [cm]입니다.
```

02-4 숫자와 문자열의 다양한 기능능

시작하기 전에

문자열의 format() 함수

문자열 형 (데이터를 보관하는 일꾼)은 format() 함수(기능)을 가지고 있습니다. format() 함수가 어떤 기능을 수행하는지 아래 예제부터 살펴 보겠습니다.

```
print( '{}'.format(24) )
print('실내 온도는 {}도 입니다.'.format(24))
```

```
24
실내 온도는 24도 입니다.
```

```
msg = '실내 온도는 {}도 입니다.'.format(24)
print( msg )
```

```
실내 온도는 24도 입니다.
```

```
1 temperature = 24
2 msg = '실내 온도는 {}도 입니다.'.format(temperature)
3 print( msg )
```

```
실내 온도는 24도 입니다.
```

```
1 print('{} + {} = {}'.format(2, 3, 5))
```

```
2 + 3 = 5
```

```
1 print('{}의 높이는 {}[m]입니다.'.format('설악산', 1188))
```

```
설악산의 높이는 1188[m]입니다.
```

IndexError 예외

{} 기호의 개수가 format() 함수의 매개변수 개수보다 많으면 IndexError 예외가 발생.

```
1 print('{} x {} = 4 '.format(2, 2) )
```

```
2 x 2 = 4
```

```
1 print('{} x {} = 4 '.format(2, 2, 2*2) )
```

```
2 x 2 = 4
```

```
1 print('{} x {} = {} '.format(2, 2) )
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-15-efbc8cfe9f17> in <cell line: 1>()
----> 1 print('{} x {} = {} '.format(2, 2) )

IndexError: Replacement index 2 out of range for positional args tuple

SEARCH STACK OVERFLOW
```

```
1 print('{} {} {} {} } '.format(0, 1, 2) )
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-16-83ef27c8008a> in <cell line: 1>()
----> 1 print('{} {} {} {} } '.format(0, 1, 2) )

IndexError: Replacement index 3 out of range for positional args tuple

SEARCH STACK OVERFLOW
```

format() 함수의 다양한 기능

정수 출력의 다양한 형태

format() 함수를 사용하여 다양한 잔재주를 부릴 수 있습니다.

```
1 print('{:d}'.format(34))
```

```
34
```

'{:d}' 의 의미는 정수 형 데이터를 출력하겠다고 명시하는 것입니다. 따라서 '{:d}' 와 같은 형식을 사용했을 때는 출력할 데이터가 정수 형이어야 합니다. 그렇다면 아래와 같은 코드를 실행시키면 어떻게 될까요? 직접 실행시켜 보시죠.

```
print('{:d}'.format(3.14))
```

```
1 print('{:d}'.format(3.14))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-ebf2ef9cb297> in <cell line: 1>()
----> 1 print('{:d}'.format(3.14))

ValueError: Unknown format code 'd' for object of type 'float'

SEARCH STACK OVERFLOW
```

'float' 형(type) 데이터를 처리하는 일꾼(object를 의역)에게 알져지지 않은 형식 코드 'd'라는 메시지와 함께 ValueError가 났다고 알려주고 있습니다. 다시 말해서 인터프리터는 float 형 데이터에 형식을 지정하는 코드 d를 모르겠다라고 '나 일 못해~' 투덜대는 겁니다. 그래서 결론은 format()함수를 쓰면서 '{:d}' 와 같은 형식을 사용했을 때는 출력할 데이터가 정수 형이어야 한다는 것입니다.

```
1 print('{:5d}'.format(34))
2 print('{:10d}'.format(34))
```

```
   34
        34
```

위 코드셀에서 '{:5d}' 의 의미는 "출력 공간을 5칸 잡고 출력할 데이터 마지막 숫자가 확보한 5칸 중 맨 뒤(다섯 번째)칸에 오도록 출력하라는 의미입니다. 아래 코드를 다시 보시죠.

```
print('{:3d}'.format(123))
print('{:5d}'.format(123))
```

```
123
  123
```

아래 코드는 어떤 반응을 할까요?

```
print('{:05d}'.format(34))
print('{:05d}'.format(-34))
```

'{:05d}' 는 "정수를 출력할 것이고 5칸을 공간에 뒤에서부터 출력할 숫자를 채우고 남는 공간은 0으로 채워라"라는 뜻입니다.

```
print('{:05d}'.format(34))
print('{:05d}'.format(-34))
```

```
00034
-0034
```

다음으로는 format() 함수를 이용하여 숫자가 출력 될 때 부호(+, -)를 붙여서 출력할 수 있다는 것을 확인하겠습니다.

```
msg_format = '{:+d}'.format(34)
print(msg_format)

msg_format = '{:+d}'.format(-34)
print(msg_format)

msg_format = '{:d}'.format(-34)
print(msg_format)
```

```
+34
-34
-34
```

```
msg_format = '{:d}'.format(34)
print(msg_format)

msg_format = '{: d}'.format(34)     # 양수 부호를 스페이스(공백)으로 대체
print(msg_format)

msg_format = '{: d}'.format(-34)
print(msg_format)
```

```
34
 34
-34
```

```
print('{:=+5d}'.format(34))
print('{:=+5d}'.format(-34))
```

```
+  34
-  34
```

▼ ◂참고▸

아래 코드의 실행 결과는 어떻게 될까요?

```
print('{:3d}'.format(1234))
```

```
print('{:3d}'.format(123))
print('{:3d}'.format(1234))
```

```
 123
1234
```

```
print('{:03d}'.format(123))
print('{:03d}'.format(1234))
```

```
123
1234
```

▼ 부동 소수점 출력의 다양한 형태

이번에는 소수점이 있는 float 형 숫자를 format() 함수를 사용하여 다양하게 표현하는 방법을 살펴보겠습니다.

```
output = '{:f}'.format(3.1415)    # 기본적인 출력 형태
print(output)

output = '{:8f}'.format(3.1415)  # 8칸 공간을 잡고 출력하되 마지막 칸부터 숫자를 채워나감.
print(output)                    # "3.141500"은 점('.')을 포함하여 8 자임(3, ., 1, 4, 1, 5, 0, 0

output = '{:9f}'.format(3.1415)  # 9칸 공간을 잡고 출력하되 마지막 칸부터 숫자를 채워나감
print(output)
```

```
3.141500
3.141500
 3.141500
```

```
output = '{:10.2f}'.format(3.1415)    # 10칸에 출력하되 뒤에서부터 채우고 소수점 이하 두 째자리까
print(output)

output = '{:10.2f}'.format(34.1651)    # 10칸에 출력하되 뒤에서부터 채우고 소수점 이하 두 째자리
print(output)
```

```
 3.14
34.17
```

기본 형태로 출력하되, 소수점 몇 째자리까지만 출력할 것인지를 정해서 화면에 출력하는 게 가장 단순하고 자주 쓰이게 형태입니다.

```
1 output = '{:f}'.format(34.1651)
2 print(output)
3
4 output = '{:.2f}'.format(34.1651)
5 print(output)
```

```
34.165100
34.17
```

더블클릭 또는 Enter 키를 눌러 수정

의미 없는 소수점 제거하기

```
1 income_per_month = 700.00
2 print('{:g}'.format(income_per_month))
```

```
700
```

대분자 바꾸기 : upper()와 lower()

```
1 name = 'Sung-pil, Kim'
2 print( name.upper() )
```

```
SUNG-PIL, KIM
```

```
1 name = 'Sung-pil, Kim'
2 print( name.lower() )
```

```
sung-pil, kim
```

중요 아래 코드를 실행 했을 때 어떤 결과가 예상됩니까?

```
name = 'Sung-pil, Kim'
print( name.upper() )
print( name )
```

실행시켜 보니 여러분이 예상한 결과와 같나요?

```
1 name = 'Sung-pil, Kim'
2 print( name.upper() )
3 print( name )
```

```
SUNG-PIL, KIM
Sung-pil, Kim
```

upper() 함수와 lower() 함수를 사용하면 name 이 가지고 있는 문자열의 내용이 바뀌는 것이 아니라 바뀐 내용만 반환하고 원 데이터는 그대로 유지됩니다. 이렇게 원 데이터가 변경되지 않는 함수를 비파괴적 함수라고 부릅니다.

"그럼 파괴적 함수도 있나요?" 당장 물어보고 싶으신 분도 있으시겠습니다만 파괴적 함수는 때가 되면 만날 겁니다.

문자열 양옆의 공백 제거하기:strip()

다루는 데이터가 대부분 숫자라면 strip() 함수를 쓸 일이 많다고 할 수는 없겠습니다. strip() 함수가 하는 기능은 문자열 앞뒤에 붙은 공백(스페이스)를 제거 합니다.

```
1 price_product = '  125,000원'
2 print( price_product )
3 print( price_product.strip() )
4
5 print( price_product )          # strip()는 비파괴적 함수라는 것을 확인할 수 있음.
```

```
  125,000원
125,000원
  125,000원
```

- strip():
- lstrip():
- rstrip():

```
1 first_name = '   Sung-pil   '
2 surname = 'Kim'
3
4 print(first_name + surname)
5 print(first_name.rstrip() + ' ' + surname)
```

```
   Sung-pil   Kim
   Sung-pil Kim
```

문자열의 구성 파악하기:isXX()

- isalnum() : 문자열이 알파벳 또는 숫자로만 구성됭 있는지 확인
- isalpha() : 문자열이 알파벳으로만 구성되어 있는지 확인
- isidentifier() : 문자열이 식별자로 사용할 수 있는지 확인

- isdecimal() : 문자열이 정수 형태인지 확인
- isdigit() : 문자열이 숫자를 나타내는 문자로만 구성되었는지 확인, 다른 말로 문자가 하나라도 있으면 False 반환

등등

```
print( "CE201".isalnum() )
```

```
True
```

```
print( "CE-201".isalnum() )
```

```
False
```

```
print( "23".isdigit() )
print( "23.12".isdigit() )    # 점('.')은 숫자를 나타내는 문자가 아님
print( "-23".isdigit() )    # 음수 부호('-')는 숫자를 나타내는 문자가 아님

```

```
True
False
False
```

문자열 찾기: find()와 rfind()

`find()`와 `rfind()` 함수는 주어진 문자열에서 특정 문자(열)과 일치하는 문자(열)의 첫 번째 인덱스 값을 반환

- `find(문자열)` : 문자열의 왼쪽부터 검색
- `rfind(문자열)`: 문자열의 오른쪽부터 검색(뒤에서부터 검색)

```
s = '가나다라가나다라'          # 문자열 s는 '가', '나', '다', '라'를 각각 2개씩 가지고 있음.
print( s.find('가') )
print( s.find('가나') )
print( s.find('나다라') )
```

```
0
0
1
```

```
s = '가나다라가나다라'          # 문자열 s는 '가', '나', '다', '라'를 각각 2개씩 가지고 있음.
print( s.rfind('가') )
print( s.rfind('가나') )
print( s.rfind('나다라') )
```

```
4
4
5
```

문자열과 in 연산자

문자열 내부에 특정 문자열이 있는지 확인하기 위한 in 연산자가 있습니다. 연산 결과는 있다 없다 둘 중에 하나겠죠? 둘 중의 하나라... 그렇다면....

```
'학교' in '종이 땡땡땡'
'종이' in '종이 땡땡땡?'
```

```
1 '학교' in '종이 땡땡때'     # 문자열, '종이 땡땡땡' 안에(in) 문자열 '학교'가 없습니다.
```

```
False
```

```
1 '종이' in '종이 땡땡땡?'    # 문자열 , '종이 땡땡때?' 안에(in) 문자열 '종이'가 있습니다.
```

```
True
```

문자열 자르기:split()

split() 함수는 문자열을 특정 문자(열)을 기준으로 나누어주는 기능을 합니다. 나누어 분리한 것들을 요소로 하는 리스트 형 테이터를 반환합니다.

```
1 your_hobby = "드라이브,낚시,등산"
2 print( your_hobby.split(',') )
```

```
['드라이브', '낚시', '등산']
```

```
1 your_hobby = "드라이브, 낚시, 등산"     # 한 문자열("드라이브, 낚시, 등산")에 각 단어는 쉼표(','
2 print( your_hobby.split(',') )
```

```
['드라이브', ' 낚시', ' 등산']
```

출력 리스트의 두 번째 요소와 세 번째 요소를 보면 '낚시'가 아니라 앞에 공백이 붙은 ' 낚시'이고 '등산'이 아니라 공백이 붙은 ' 등산'입니다. 결과가 썩 마음에 들지는 않네요. 그렇다면 아래와 같이 코드를 수정하겠습니다.

```
1 your_hobby = "드라이브, 낚시, 등산"     # 한 문자열("드라이브, 낚시, 등산")에 각 단어는 쉼표(','
2 print( your_hobby.split(', ') )
```

```
['드라이브', '낚시', '등산']
```

```
1 your_hobby = "드라이브|낚시|등산"     # 한 문자열("드라이브, 낚시, 등산")에 각 단어는 쉼표(',')요
2 print( your_hobby.split('|') )
```

```
['드라이브', '낚시', '등산']
```

◂참고▸ f-문자열

아래 코드 분석

```
1 print('3+4=' + str(3+4))
```

```
3+4=7
```

```
1 print('3+4={}'.format(3+4))
```

```
3+4=7
```

```
1 print(f'3+4={3+4}')
```

```
3+4=7
```

```
1 name = '김성필'
2 hobby = '낚시'
3 address = '경기도 화성시 동탄'
4
5 print(f'{name}의 취미는 {hobby}이고 사는 곳은 {address}입니다.')
```

```
김성필의 취미는 낚시이고 사는 곳은 경기도 화성시 동탄입니다.
```