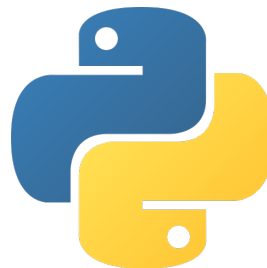




Python 프로그래밍 [2]

[자료형 / 문자열 / 변수와 입력]



SW융합학부

강희국

※ 수강생을 위한 참고자료로 제3자에 대한 배포를 금지합니다. 법적인 문제 발생 시 배포자에게 책임이 있습니다.

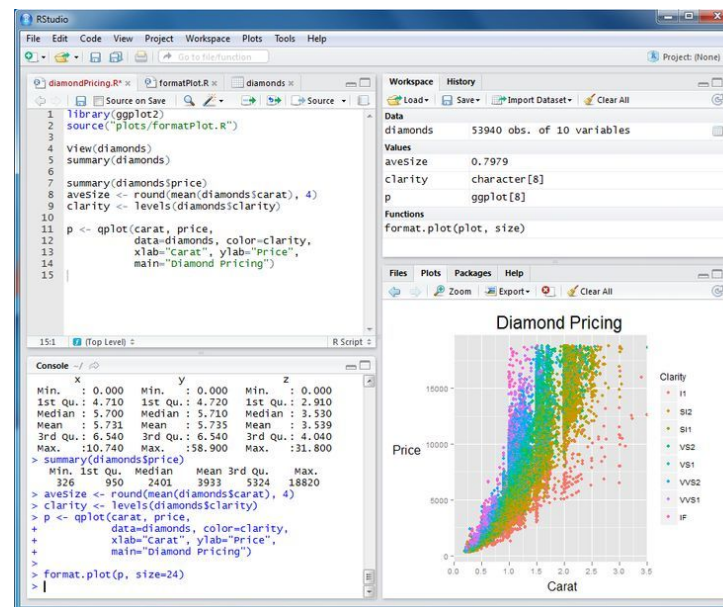
1 자료형과 문자열

2 숫자와 연산자

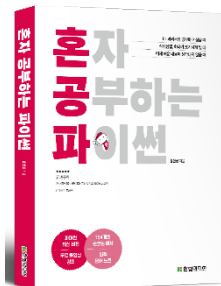
3 변수와 입력

4 숫자와 문자열의 다양한 기능

5 -



1. 자료형과 문자열



[참고] 윤인성, "혼자 공부하는 파이썬", 한빛미디어

- **자료** (data): 프로그램이 처리할 수 있는 모든 것
 - 프로그램은 자료를 처리하기 위한 모든 행위
- **자료형** (data type): 자료를 기능과 역할에 따라 구분한 것
 - **문자열** (string) : 메일 제목, 메시지 내용 등
 - **숫자** (number) : 물건의 가격, 학생의 성적 등
 - **불** (boolean) : 친구의 로그인 상태 등

● 자료형 (data type)

- 자료의 형식
- `type()` 함수로 확인

```
>>> print(type("안녕하세요"))  
<class 'str'>  
>>> print(type(273))  
<class 'int'>
```

- `str` : 문자열
- `int` : 정수

● 문자열 (string)

➤ 따옴표로 둘러싸 입력하는, 글자가 나열된 것

"Hello" 'String' '안녕하세요' "Hello Python Programming"

```
# 하나만 출력합니다.
print("# 하나만 출력합니다.")
print("Hello Python Programming...!")
print()

# 여러 개를 출력합니다.
print("# 여러 개를 출력합니다.")
print(10, 20, 30, 40, 50)
print("안녕하세요", "저의", "이름은", "윤인성입니다!")
...
```

문자열

문자열

문자열

- 큰따옴표로 문자열 만들기

```
>>> print("안녕하세요")  
안녕하세요
```

- 작은따옴표로 문자열 만들기

```
>>> print('안녕하세요')  
안녕하세요
```

● 문자열 내부에 따옴표 넣기

"안녕하세요"라고 말했습니다

출력할 큰따옴표

```
>>> print("안녕하세요"라고 말했습니다)
```

문자열을 만들기 위해 사용한 큰따옴표

위 경우 오류 발생

- 파이썬 프로그래밍 언어는 자료와 자료를 단순 나열할 수 없음
- **구문 오류** (syntax error)

오류

SyntaxError: invalid syntax

- 작은따옴표로 문자열 만들어 큰따옴표 포함 문제 해결
 - 반대로도 가능

```
>>> print('"안녕하세요"라고 말했습니다')  
"안녕하세요"라고 말했습니다
```

```
>>> print("'배가 고픈다'라고 생각했습니다")  
'배가 고픈다'라고 생각했습니다
```

● 이스케이프 문자 (escape character)

- 역슬래시 기호와 함께 조합해서 사용하는 특수한 문자
- \“ : 큰따옴표를 의미
- \‘ : 작은따옴표를 의미

```
>>> print("\"안녕하세요\"라고 말했습니다")
"안녕하세요"라고 말했습니다
>>> print('\배가 고픈다\'라고 생각했습니다')
'배가 고픈다'라고 생각했습니다
```

- \n : 줄바꿈 의미
- \t : 탭 의미

```
>>> print("안녕하세요\n안녕하세요")
```

```
안녕하세요
```

```
안녕하세요
```

```
>>> print("안녕하세요\t안녕하세요")
```

```
안녕하세요      안녕하세요
```

```
01 print("이름\t나이\t지역")
```

```
02 print("윤인성\t25\t강서구")
```

```
03 print("윤아린\t24\t강서구")
```

```
04 print("구름\t3\t강서구")
```

실행결과		
이름	나이	지역
윤인성	25	강서구
윤아린	24	강서구
구름	3	강서구

➤ \\: 역슬래시를 의미

```
>>> print("\\ \\ \\ \\")  
\\ \\
```

● 여러 줄 문자열 만들기

➤ \n 사용

```
>>> print("동해물과 백두산이 마르고 닳도록\n하느님이 보우하사 우리나라 만세\n무궁화 삼천리 화려강\n산 대한사람\n대한으로 길이 보전하세")  
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

- 여러 줄 문자열 기능 활용: 큰따옴표 혹은 작은따옴표를 세 번 반복

```
>>> print("""동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세""")  
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

● 줄바꿈 없이 문자열 만들기

➤ \ 기호 사용

```
>>> print("""  
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세  
""")
```

```
동해물과 백두산이 마르고 닳도록  
하느님이 보우하사 우리나라 만세  
무궁화 삼천리 화려강산 대한사람  
대한으로 길이 보전하세
```

→ 위 아래로 의도하지 않은 줄바꿈이 들어갑니다.

- 줄 뒤에 \ 붙여서 코드 쉽게 보기 위한 줄바꿈이며, 실질적 줄바꿈 아님을 나타냄

```
>>> print("""\
```

```
동해물과 백두산이 마르고 닳도록
```

```
하느님이 보우하사 우리나라 만세
```

```
무궁화 삼천리 화려강산 대한사람
```

```
대한으로 길이 보전하세\
```

```
""")
```

```
동해물과 백두산이 마르고 닳도록
```

```
하느님이 보우하사 우리나라 만세
```

```
무궁화 삼천리 화려강산 대한사람
```

```
대한으로 길이 보전하세
```

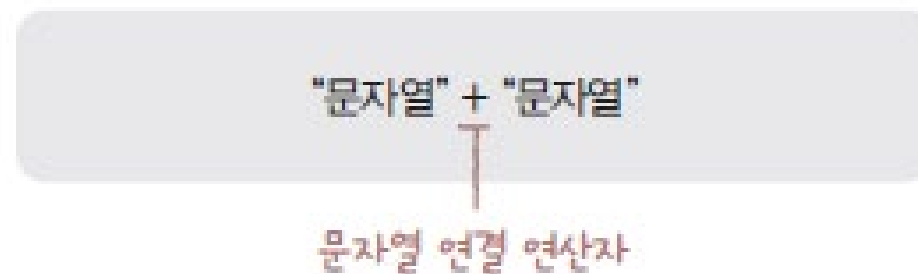
→ 줄을 바꿔 출력하지 않겠다고 선언합니다.

- 숫자에는 사칙연산 연산자를, 집합에는 여러 집합 연산자 적용 가능

➤ 각 자료는 사용할 수 있는 연산자 정해져 있음



- 문자열 연결 연산자 : +




➤ 더하기와 같은 기호이나 다른 수행임에 주의

- 두 문자열 연결하여 새로운 문자열 만들어냄

```
>>> print("안녕" + "하세요")
안녕하세요
>>> print("안녕하세요" + "!")
안녕하세요!
```

- 문자열과 숫자 사이에는 사용할 수 없음

```
>>> print("안녕하세요" + 1)
```

 오류

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

- 문자열은 문자끼리, 숫자는 숫자끼리 연결
- 문자열과 숫자 연결하여 연산하려면 큰따옴표 붙여
문자열로 인식하게 함

● 문자열 반복 연산자 : *

➤ 문자열을 숫자와 * 연산자로 연결

```
>>> print("안녕하세요" * 3)  
안녕하세요안녕하세요안녕하세요
```

```
>>> print(3 * "안녕하세요")  
안녕하세요안녕하세요안녕하세요
```

● 문자 선택 연산자 (인덱싱) : []

- 문자열 내부의 문자 하나를 선택
- 대괄호 안에 선택할 문자의 위치를 지정
- **인덱스** (index)
 - **제로 인덱스** (zero index) : 숫자를 0부터 셈
 - **원 인덱스** (one index) : 숫자를 1부터 셈
 - 파이썬은 제로 인덱스 유형 사용

안	녕	하	세	요
[0]	[1]	[2]	[3]	[4]

➤ 예시

```
01 print("문자 선택 연산자에 대해 알아보까요?")
02 print("안녕하세요"[0])
03 print("안녕하세요"[1])
04 print("안녕하세요"[2])
05 print("안녕하세요"[3])
06 print("안녕하세요"[4])
```

실행결과

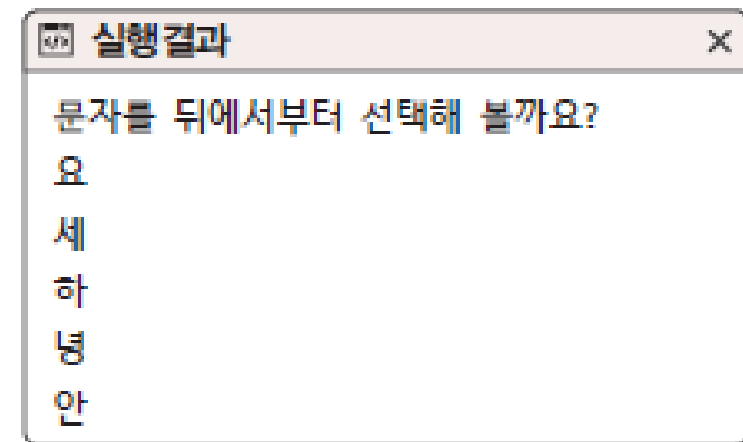
문자 선택 연산자에 대해 알아보까요?
안
녕
하
세
요

➤ 문자를 거꾸로 출력하려는 경우

- 대괄호 안 숫자를 음수로 입력

안	녕	하	세	요
[-5]	[-4]	[-3]	[-2]	[-1]

```
01 print("문자를 뒤에서부터 선택해 볼까요?")
02 print("안녕하세요"[-1])
03 print("안녕하세요"[-2])
04 print("안녕하세요"[-3])
05 print("안녕하세요"[-4])
06 print("안녕하세요"[-5])
```



● 문자열 범위 선택 연산자 (슬라이싱) : [:]

- 문자열의 특정 범위를 선택
- 대괄호 안에 범위 구분 위치를 콜론으로 구분

```
>>> print("안녕하세요"[1:4])  
녕하세
```

- 마지막 숫자 포함
- 마지막 숫자 포함하지 않음
 - 파이썬에서 적용

안	녕	하	세	요
[0]	[1]	[2]	[3]	[4]

➤ 예시

```
>>> print("안녕하세요"[0:2])
안녕
>>> print("안녕하세요"[1:3])
녕하
>>> print("안녕하세요"[2:4])
하세
```

➤ 대괄호 안에 넣는 숫자 둘 중 하나를 생략하는 경우

- 뒤의 값 생략 : n번째부터 끝의 문자까지
- 앞의 값 생략 : 0번째부터 뒤의 숫자 n번째 앞의 문자까지

[1:]

[:3]

```
>>> print("안녕하세요"[1:])
녕하세요
>>> print("안녕하세요"[:3])
안녕하
```

- **인덱싱** (indexing)

- [] 기호 이용해 문자열의 특정 위치에 있는 문자 참조하는 것

- **슬라이싱** (slicing)

- [:] 기호 이용해 문자열 일부를 추출하는 것
- 문자열 선택 연산자로 슬라이스해도 원본은 변하지 않음에 주의

```
>>> hello = "안녕하세요" → ①
>>> print(hello[0:2]) → ②
안녕
>>> hello → ③
'안녕하세요'
```


● IndexError (index out of range) 예외

- 리스트/문자열 수를 넘는 요소/글자 선택할 경우 발생

```
>>> print("안녕하세요"[10])
```

❗ 오류

→ 파이썬 IDLE 에디터에서 실행했을 때 나타나는 내용으로 에디터마다 다르게 나타납니다.

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

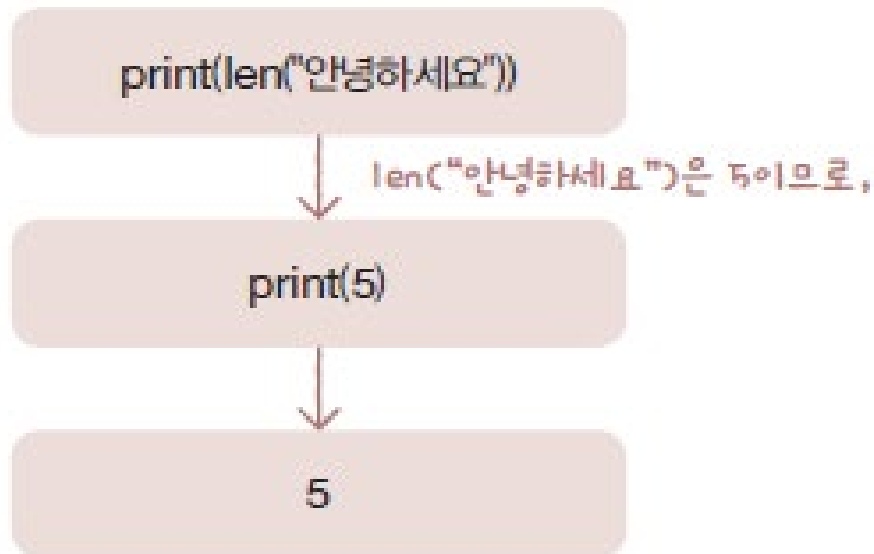
print("안녕하세요"[10])

IndexError: string index out of range → IndexError 예외가 발생했어요.

● len() 함수

- 문자열 길이 구할 때 사용
- 괄호 내부에 문자열 넣으면 문자열의 문자 개수 세어 줌
- 중첩된 구조의 함수는 괄호 안쪽부터 먼저 실행

```
>>> print(len("안녕하세요"))  
5
```



- **자료형** : 자료의 형식
- **문자열** : 문자의 나열. 큰따옴표 혹은 작은따옴표로 입력
- **이스케이프 문자** : 문자열 내부에서 특수한 기능 수행하는 문자열
- **문자열 연산자**

문자열 연결 연산자 (+), 문자열 반복 연산자 (*), 문자열 선택 연산자 ([]),
문자열 범위 선택 연산자 ([:])

- **type()** : 자료형 확인하는 함수
- **len()** : 문자열 길이 구하는 함수

- 문자열을 만드는 파이썬 구문의 빈칸에 알맞은 기호를 넣어보세요.

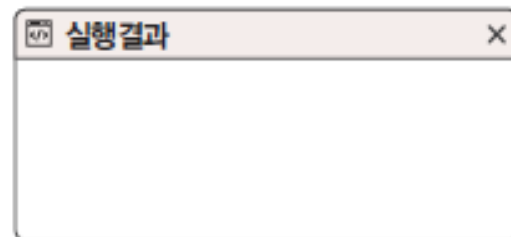
구문	의미
<code>_____ 문자 _____</code>	큰따옴표로 문자열 만들기
<code>_____ 문자 _____</code>	작은따옴표로 문자열 만들기
<code>_____ 문자열 문자열 문자열 _____</code>	여러 문자열 만들기

- 이스케이프 문자의 의미를 보고 알맞은 기호 혹은 문자를 넣어보세요.

이스케이프 문자	의미
<code>_____</code>	큰따옴표를 의미합니다.
<code>_____</code>	작은따옴표를 의미합니다.
<code>_____</code>	줄바꿈을 의미합니다.
<code>_____</code>	탭을 의미합니다.
<code>_____</code>	\을 의미합니다.

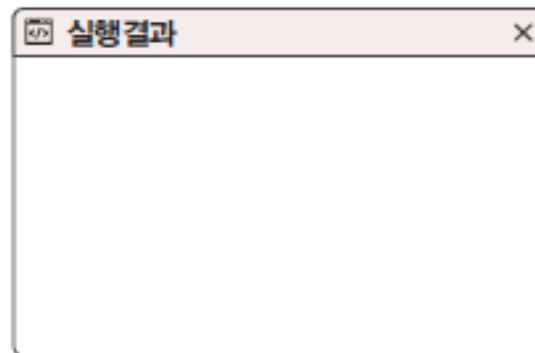
- 다음 프로그램의 실행결과를 예측해보세요.

```
print("# 연습 문제")  
print("\\\\\\\\\\\\\\\\")  
print("-" * 8)
```



- 다음 프로그램의 실행결과를 예측해보세요. 오류가 발생하는 것은 어느 행 인가요? 그리고 그 이유는 무엇인가요?

```
print("안녕하세요"[1])  
print("안녕하세요"[2])  
print("안녕하세요"[3])  
print("안녕하세요"[4])  
print("안녕하세요"[5])
```



2. 숫자와 연산자



[참고] 윤인성, “혼자 공부하는 파이썬”, 한빛미디어

● 정수형

- 소수점이 없는 숫자
- 0, 1, 273, -52
- 정수 (integer)

● 실수형

- 소수점이 있는 숫자
- 0.0, 52.273, -1.2
- 실수 (floating point, 부동 소수점)

- 숫자를 만들기 위해서는 단순히 숫자 입력하면 됨

```
>>> print(273)
273
>>> print(52.273)
52.273
```

- type() 함수로 소수점 없는 숫자와 있는 숫자를 출력

```
>>> print(type(52))
<class 'int'>
>>> print(type(52.273))
<class 'float'>
```

- Int : 정수
- Float : 부동 소수점 (실수)
 - 일반적으로 프로그래밍 언어에서는 두 자료형을 구분해서 사용

● 사칙 연산자 : +, -, *, /

➤ 덧셈, 뺄셈, 곱셈, 나눗셈

연산자	설명	구문	연산자	설명	구문
+	덧셈 연산자	숫자 + 숫자	*	곱셈 연산자	숫자 * 숫자
-	뺄셈 연산자	숫자 - 숫자	/	나눗셈 연산자	숫자 / 숫자

```
>>> print("5 + 7 =", 5 + 7)
5 + 7 = 12
>>> print("5 - 7 =", 5 - 7)
5 - 7 = -2
>>> print("5 * 7 =", 5 * 7)
5 * 7 = 35
>>> print("5 / 7 =", 5 / 7)
5 / 7 = 0.7142857142857143
```

● 정수 나누기 연산자: //

- 숫자를 나누고 소수점 이하 자릿수 삭제한 후 정수 부분만 남김

```
>>> print("3 / 2 =", 3 / 2)
3 / 2 = 1.5
>>> print("3 // 2 =", 3 // 2)
3 // 2 = 1
```

● 나머지 연산자 : %

- A를 B로 나누었을 때의 나머지를 구함

```
>>> print("5 % 2 =", 5 % 2)
5 % 2 = 1
```

● 제공 연산자 : **

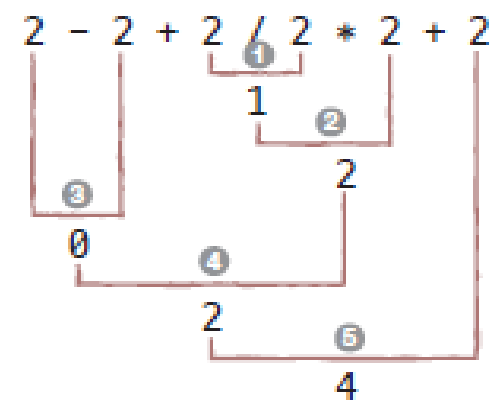
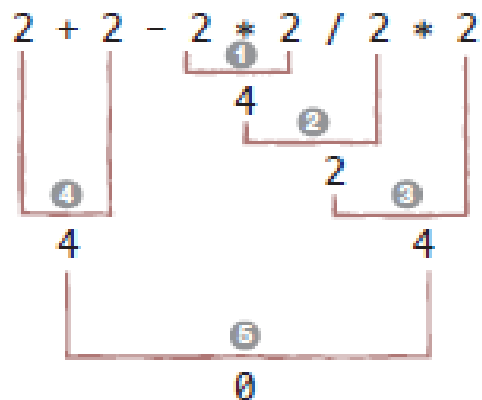
➤ 숫자를 제곱함

```
>>> print("2 ** 1 =", 2 ** 1)
2 ** 1 = 2
>>> print("2 ** 2 =", 2 ** 2)
2 ** 2 = 4
>>> print("2 ** 3 =", 2 ** 3)
2 ** 3 = 8
>>> print("2 ** 4 =", 2 ** 4)
2 ** 4 = 16
```

● 우선순위

- 파이썬의 수식은 연산자 간 우선순위에 따라 계산됨
 - 곱셈과 나눗셈이 덧셈과 뺄셈보다 우선

```
>>> print(2 + 2 - 2 * 2 / 2 * 2)
0.0
>>> print(2 - 2 + 2 / 2 * 2 + 2)
4.0
```



- 괄호 활용하여 우선순위 조정

$$(5 + 3) * 2$$

- 연산자 우선순위 확실한 경우에도 괄호로 감싸는 것이 좋음

$$5 + (3 * 2)$$

● TypeError 예외

➤ 서로 다른 자료를 연산할 경우

```
>>> string = "문자열"  
>>> number = 273  
>>> string + number
```

[오류]

Traceback (most recent call last):

File "<pyshell#3>", line 1, in <module>

string+number

TypeError: can only concatenate str (not "int") to str → TypeError 예외가 발생했어요.

● 숫자 자료형

- 소수점이 없는 정수형과 소수점이 있는 실수형 (부동 소수점)

● 숫자 연산자

- 사칙연산자와 // (정수 나누기 연산자), % (나누기 연산자), ** (제곱 연산자) 등

● 연산자

- **우선순위**가 존재하는데, 곱하기와 나누기가 가장 우선이고 더하기와 빼기가 다음으로, 잘 모를 때는 괄호를 입력

- 오른쪽의 예시를 보고 숫자 자료형을 나타내는 단어를 쓰세요.

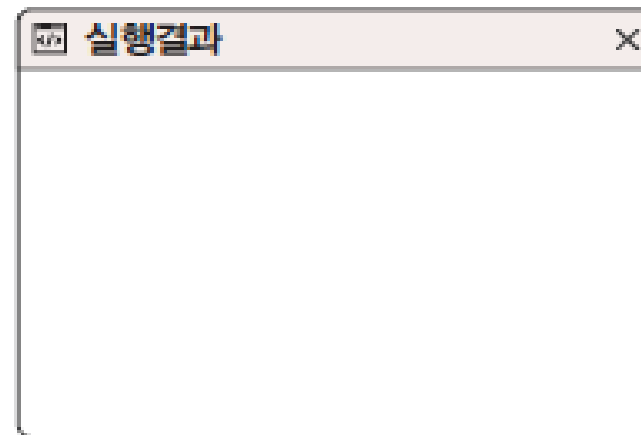
단어	예시
<input type="text"/>	273, 52, 0, 1234, -25
<input type="text"/>	0.0, 1.234, 2.73e2, -25.0

- 숫자에 적용할 수 있는 연산자입니다.
의미를 보고 왼쪽 연산자 항목에 기호를 써 보세요.

연산자	의미
<input type="text"/>	덧셈 연산자
<input type="text"/>	뺄셈 연산자
<input type="text"/>	곱셈 연산자
<input type="text"/>	나눗셈 연산자
<input type="text"/>	정수 나누기 연산자
<input type="text"/>	나머지 연산자
<input type="text"/>	제곱 연산자

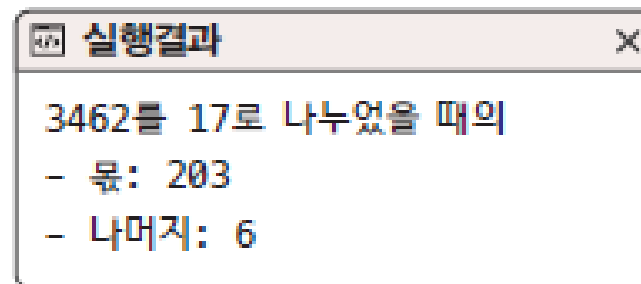
- 다음 프로그램의 실행결과를 예측해 보세요.

```
print("# 기본적인 연산")  
print(15, "+", 4, "=", 15 + 4)  
print(15, "-", 4, "=", 15 - 4)  
print(15, "*", 4, "=", 15 * 4)  
print(15, "/", 4, "=", 15 / 4)
```



- 3472를 17로 나누었을 때의 몫과 나머지를 구하는 프로그램입니다. 빈칸을 채워 완성해 주세요.

```
print("3462를 17로 나누었을 때의")  
print("- 몫:", )  
print("- 나머지:", )
```



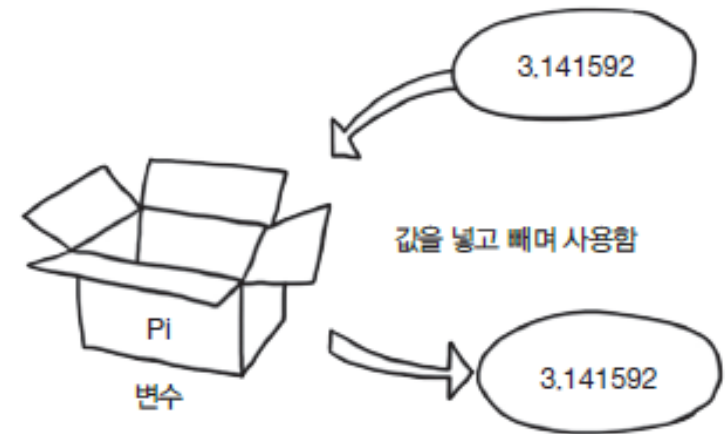
3. 변수와 입력



● 변수

- 값을 저장할 때 사용하는 식별자
- 숫자뿐만 아니라 모든 자료형을 저장할 수 있음

```
>>> pi = 3.14159265  
>>> pi  
3.14159265
```



● 변수의 활용

- 변수를 선언하는 방법
 - 변수를 생성
- 변수에 값을 할당하는 방법
 - 변수에 값을 넣음
 - = 우변을 값을 좌변에 할당
- 변수를 참조하는 방법
 - 변수에서 값을 꺼냄
 - 변수 안에 있는 값을 사용

변수 = 값

값을 변수에 할당합니다.

● 변수를 참조

- 변수에 저장된 값을 출력

```
변수
```

- 변수에 저장된 값으로 연산

```
변수 + 변수
```

- 변수에 저장된 값을 출력

```
print(변수)
```

- 앞 예시에서 입력한 pi는 숫자 자료에 이름 붙인 것이기 때문에 숫자 연산 모두 수행할 수 있음

```
>>> pi = 3.14159265
>>> pi + 2
5.14159265
>>> pi - 2
1.1415926500000002
>>> pi * 2
6.2831853
>>> pi / 2
1.570796325
>>> pi % 2
1.1415926500000002
>>> pi * pi
9.869604378534024
```

- pi는 숫자 자료이므로 숫자와 문자열 연산은 불가능

```
pi + "문자열"
```

- 예시 - 원의 둘레와 넓이 구하기

```
01  # 변수 선언과 할당
02  pi = 3.14159265
03  r = 10
04
05  # 변수 참조
06  print("원주율 =", pi)
07  print("반지름 =", r)
08  print("원의 둘레 =", 2*pi*r)      # 원의 둘레
09  print("원의 넓이 =", pi*r*r)     # 원의 넓이
```

실행결과

```
원주율 = 3.14159265
반지름 = 10
원의 둘레 = 62.831853
원의 넓이 = 314.159265
```

● 복합 대입 연산자

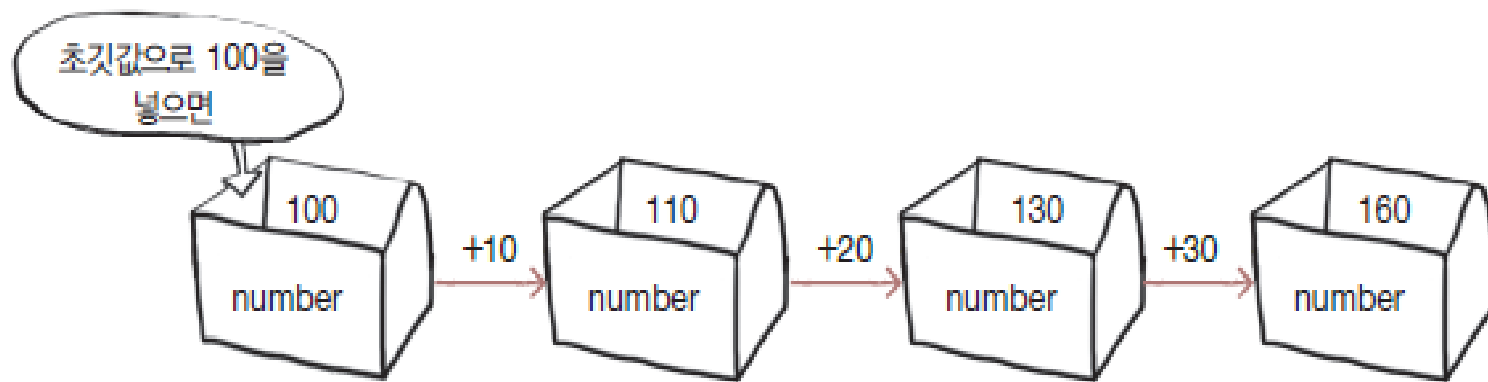
➤ 기본 연산자와 = 연산자 함께 사용해 구성

```
a += 10
```

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

➤ 예시

```
>>> number = 100  
>>> number += 10  
>>> number += 20  
>>> number += 30  
>>> print("number:", number)  
number: 160
```

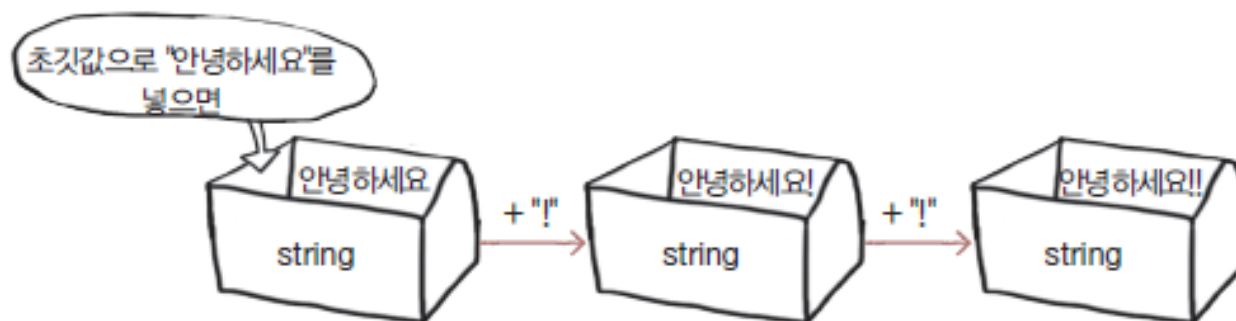


➤ 문자열 역시 복합 대입 연산자 사용 가능

연산자 이름	설명
<code>+=</code>	문자열 연결 후 대입
<code>*=</code>	문자열 반복 후 대입

➤ 예시

```
>>> string = "안녕하세요"
>>> string += "!"
>>> string += "!"
>>> print("string:", string)
string: 안녕하세요!!
```



● input() 함수

- 명령 프롬프트에서 사용자로부터 데이터 입력받을 때 사용

● input() 함수로 사용자 입력받기

- 프롬프트 함수 : input 함수 괄호 안에 입력한 내용

```
>>> input("인사말을 입력하세요> ")
```

- 블록 (block) : 프로그램이 실행 중 잠시 멈추는 것

인사말을 입력하세요> | → 입력 대기를 알려주는 커서입니다. 커서는 프로그램에 따라 모양이 다를 수 있습니다.

- 명령 프롬프트에서 글자 입력 후 [Enter] 클릭

```
인사말을 입력하세요> 안녕하세요 [Enter]  
'안녕하세요'
```

- input 함수의 결과로 산출 (리턴값)
 - 다른 변수에 대입하여 사용 가능

```
>>> string = input("인사말을 입력하세요> ")
인사말을 입력하세요> 안녕하세요 Enter
>>> print(string)
안녕하세요
```

- input() 함수의 입력 자료형

- type() 함수로 자료형 알아보기

```
>>> print(type(string))  
<class 'str'>
```

```
>>> number = input("숫자를 입력하세요> ")  
숫자를 입력하세요> 12345   
>>> print(number)  
12345
```

```
>>> print(type(number))  
<class 'str'>
```

- input() 함수는 사용자가 무엇을 입력해도 결과는 무조건 문자열 자료형

➤ 예시 - 입력 자료형 확인하기

```
01  # 입력을 받습니다.  
02  string = input("입력> ")  
03  
04  # 출력합니다.  
05  print("자료:", string)  
06  print("자료형:", type(string))
```

실행결과 1

입력> 52273

자료: 52273

자료형: <class 'str'>

실행결과 2

입력> True

자료: True

자료형: <class 'str'>

➤ 예시 - 입력받고 더하기

```
01  # 입력을 받습니다.  
02  string = input("입력> ")  
03  
04  # 출력합니다.  
05  print("입력 + 100:", string + 100)
```

실행결과

입력> 300

Traceback (most recent call last):

File "inputerror.py", line 5, in <module>

print("입력 + 100:", string + 100)

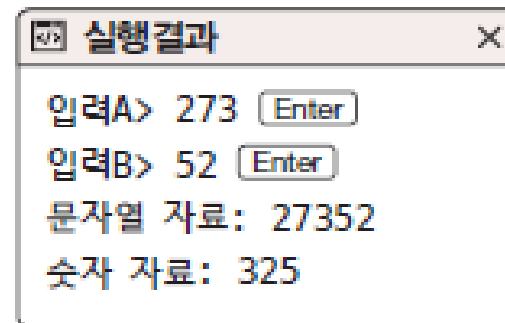
TypeError: can only concatenate str (not "int") to str

● 캐스트 (cast)

- input() 함수의 입력 자료형은 항상 문자열이므로 입력받은 문자열을 숫자 연산에 활용하기 위해 숫자로 변환
- int() 함수
 - 문자열을 int 자료형으로 변환.
- float() 함수
 - 문자열을 float 자료형으로 변환

➤ 예시 - int() 함수 활용하기

```
01 string_a = input("입력A> ")
02 int_a = int(string_a)
03
04 string_b = input("입력B> ")
05 int_b = int(string_b)
06
07 print("문자열 자료:", string_a + string_b)
08 print("숫자 자료:", int_a + int_b)
```



➤ 예시 - int() 함수와 float() 함수 활용하기

```
01 output_a = int("52")
02 output_b = float("52.273")
03
04 print(type(output_a), output_a)
05 print(type(output_b), output_b)
```

실행결과

```
<class 'int'> 52
<class 'float'> 52.273
```

➤ 예시 - int() 함수와 float 함수 조합하기

```
01 input_a = float(input("첫 번째 숫자> "))
02 input_b = float(input("두 번째 숫자> "))
03
04 print("덧셈 결과:", input_a + input_b)
05 print("뺄셈 결과:", input_a - input_b)
06 print("곱셈 결과:", input_a * input_b)
07 print("나눗셈 결과:", input_a / input_b)
```


실행결과

```
첫 번째 숫자> 273 [Enter]
두 번째 숫자> 52 [Enter]
덧셈 결과: 325.0
뺄셈 결과: 221.0
곱셈 결과: 14196.0
나눗셈 결과: 5.25
```

● ValueError 예외

- 변환할 수 없는 것을 변환하려 할 경우
- 숫자가 아닌 것을 숫자로 변환하려 할 경우


```
int("안녕하세요")  
float("안녕하세요")
```

 오류

```
Traceback (most recent call last):  
  File "intconvert.py", line 2, in <module>  
    int_a = int(string_a)  
ValueError: invalid literal for int() with base 10: '안녕하세요'
```

- 소수점이 있는 숫자 형식의 문자열을 `int()` 함수로 변환하려 할 때

```
int("52.273")
```

 오류

```
Traceback (most recent call last):  
  File "intconvert.py", line 2, in <module>  
    int_a = int(string_a)  
ValueError: invalid literal for int() with base 10: '52.273'
```

● str() 함수

➤ 숫자를 문자열로 변환

str(다른 자료형)

```
01 output_a = str(52)
02 output_b = str(52.273)
03 print(type(output_a), output_a)
04 print(type(output_b), output_b)
```

실행결과

```
<class 'str'> 52
<class 'str'> 52.273
```

- 변수 선언 : 변수를 생성하는 것을 의미
- 변수 할당 : 변수에 값을 넣는 것을 의미
- 변수 참조 : 변수에서 값을 꺼내는 것
- input() 함수 : 명령 프롬프트에서 사용자로부터 데이터 입력 받음
- int() 함수 : 문자열을 int 자료형으로 변환
- float 함수 : 문자열을 float 자료형으로 변환
- str() 함수 : 숫자를 문자열로 변환

- 변수에 값을 할당하기 위한 구문입니다. 빈칸에 알맞은 기호를 쓰세요.

변수 이름 값

- 숫자에 적용할 수 있는 복합 대입 연산자입니다. 왼쪽 연산자 항목에 알맞은 기호를 써 보세요.

연산자	내용
<input type="text"/>	숫자 덧셈 후 대입
<input type="text"/>	숫자 뺄셈 후 대입
<input type="text"/>	숫자 곱셈 후 대입
<input type="text"/>	숫자 나눗셈 후 대입
<input type="text"/>	숫자 나머지 구한 후 대입
<input type="text"/>	숫자 제곱 후 대입

- 다음 코드는 inch 단위의 자료를 입력 받아 cm를 구하는 예제입니다. 빈칸에 알맞은 내용을 넣어 코드를 완성해 주세요. (1inch = 2.54cm)

```
str_input =  ("숫자 입력> ")  
num_input =  (str_input)  
  
print()  
print(num_input, "inch")  
print((num_input * 2.54), "cm")
```

실행결과 1

숫자 입력> 1

1.0 inch
2.54 cm

실행결과 2

숫자 입력> 26

26.0 inch
66.04 cm

- 원의 반지름을 입력 받아 원의 둘레와 넓이를 구하는 코드입니다. 빈칸에 알맞은 내용을 넣어 코드를 완성해 주세요.
 - 둘레 : $2 * \text{원주율} * \text{반지름}$
 - 넓이 : $\text{원주율} * \text{반지름} * \text{반지름}$

```
str_input =  ("원의 반지름 입력> ")
num_input =  (str_input)
print()
print("반지름: ", num_input)
print("둘레: ", 2 * 3.14 * )
print("넓이: ", 3.14 *  ** 2)
```

실행결과 1

원의 반지름 입력> 2

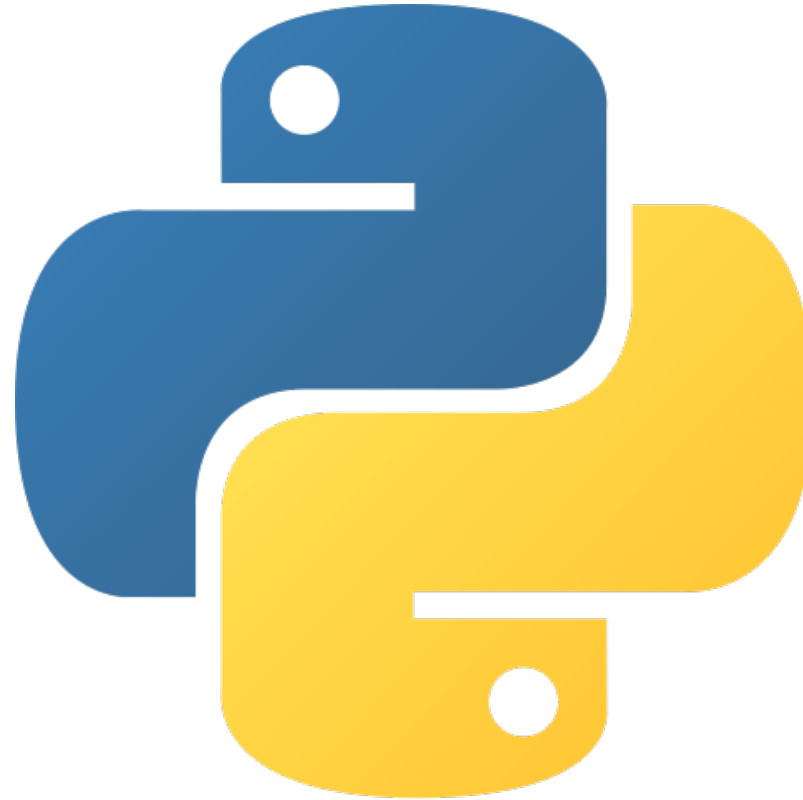
반지름: 2.0
둘레: 12.56
넓이: 12.56

실행결과 2

원의 반지름 입력> 5

반지름: 5.0
둘레: 31.400000000000002
넓이: 78.5

4. 숫자와 문자열의 다양한 기능



[참고] 윤인성, "혼자 공부하는 파이썬", 한빛미디어

[핵심 키워드] format(), upper(), lower(), strip(), find(), in연산자, split()

[핵심 포인트] 함수는 영어로 function, 즉 사람 또는 사물의 기능이라는 뜻을 가진 단어와 동음이의어다. 지금까지 살펴본 숫자나 문자열과 같은 자료도 컴퓨터에서는 하나의 사물처럼 취급되기에 내부적으로 여러 기능을 가지고 있다.

- 문자열 뒤에 마침표 입력해 보면 자동 완성 기능으로 다양한 자체 기능들이 제시됨

```
3 format_b = "파이썬 열공하여 첫 연봉 {}만 원 만들기".format(5000)
4 format_c = "{} {} {}".format(1, 2, 3)
5 format_d = "{} {} {}".format(1, 2, 3)
6 format_e = "{} {}".format(1, 2)
7 format_f = "{} {} {}".format(1, 2, 3)
8
9 # 출력하기
10 print(format_a)
11 print(format_b)
12 print(format_c)
13 print(format_d)
14 print(format_e)
15 print(format_f)
```

- capitalize
- casefold
- center
- count
- encode
- endswith
- expandtabs
- find
- format
- format_map
- index
- isalnum

● format() 함수로 숫자를 문자열로 변환

- 중괄호 포함한 문자열 뒤에 마침표 찍고 format() 함수 사용하되, 중괄호 개수와 format 함수 안 매개변수의 개수는 반드시 같아야 함
- 문자열의 중괄호 기호가 format() 함수 괄호 안의 매개변수로 차례로 대치되면서 숫자가 문자열이 됨

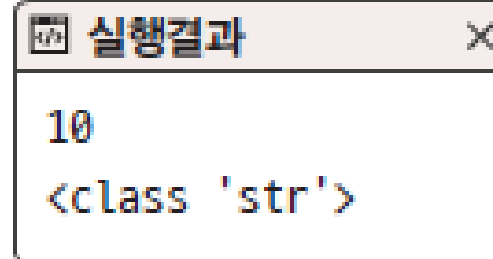
```
"{}".format(10)
```

```
"{} {}".format(10, 20)
```

```
"{} {} {} {} {}".format(101, 202, 303, 404, 505)
```

➤ 예시 - format() 함수로 숫자를 문자열로 변환하기

```
01  # format() 함수로 숫자를 문자열로 변환하기
02  string_a = "{}".format(10)
03
04  # 출력하기
05  print(string_a)
06  print(type(string_a))
```



실행결과

10
<class 'str'>

➤ 예시 - format() 함수의 다양한 형태

```
01  # format() 함수로 숫자를 문자열로 변환하기
02  format_a = "{}만 원".format(5000)
03  format_b = "파이썬 열공하여 첫 연봉 {}만 원 만들기 ".format(5000)
04  format_c = "{} {} {}".format(3000, 4000, 5000)
05  format_d = "{} {} {}".format(1, "문자열", True)
06
07  # 출력하기
08  print(format_a)
09  print(format_b)
10  print(format_c)
11  print(format_d)
```

실행결과

```
5000만 원
파이썬 열공하여 첫 연봉 5000만 원 만들기
3000 4000 5000
1 문자열 True
```

- format_a : 중괄호 옆에 다른 문자열 넣음
- format_b : 중괄호 앞뒤로 다른 문자열 넣음
- format_c : 매개변수 여러 개 넣음

● IndexError 예외

- 중괄호 기호의 개수가 format() 함수의 매개변수 개수보다 많은 경우

```
>>> "{} {}".format(1, 2, 3, 4, 5)
'1 2'

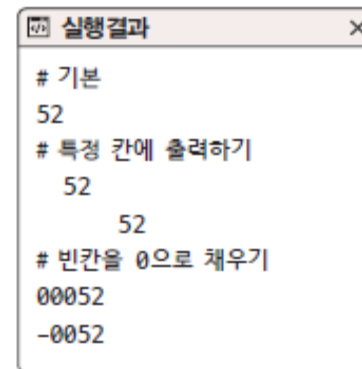
>>> "{} {} {}".format(1, 2)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    "{} {} {}".format(1, 2)
IndexError: tuple index out of range
```


● 정수 출력의 다양한 형태

➤ 예시 - 정수를 특정 칸에 출력하기

```
01  # 정수
02  output_a = "{:d}".format(52)
03
04  # 특정 칸에 출력하기
05  output_b = "{:5d}".format(52)      # 5칸
06  output_c = "{:10d}".format(52)     # 10칸
07
08  # 빈칸을 0으로 채우기
09  output_d = "{:05d}".format(52)     # 양수
10  output_e = "{:05d}".format(-52)    # 음수
11
12  print("# 기본")
13  print(output_a)
14  print("# 특정 칸에 출력하기")
15  print(output_b)
16  print(output_c)
17  print("# 빈칸을 0으로 채우기")
18  print(output_d)
19  print(output_e)
```

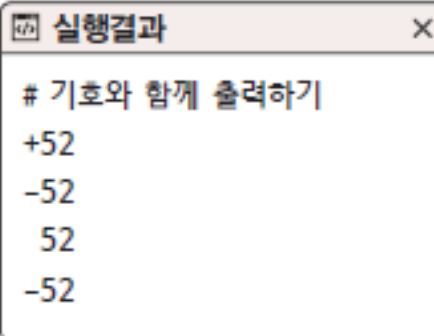
output_a : {:d}를 사용하여 int 자료형 정수 출력한다는 것을 직접 지정
output_b, output_c : 특정 칸에 맞춰서 숫자를 출력하는 형태
output_d, output_e : 빈칸을 0으로 채우는 형태



```
# 실행결과
# 기본
52
# 특정 칸에 출력하기
    52
      52
# 빈칸을 0으로 채우기
00052
-0052
```

➤ 예시 - 기호 붙여 출력하기

```
01  # 기호와 함께 출력하기
02  output_f = "{:+d}".format(52)  # 양수
03  output_g = "{:+d}".format(-52) # 음수
04  output_h = "{: d}".format(52)  # 양수: 기호 부분 공백
05  output_i = "{: d}".format(-52) # 음수: 기호 부분 공백
06
07  print("# 기호와 함께 출력하기")
08  print(output_f)
09  print(output_g)
10  print(output_h)
11  print(output_i)
```



실행결과

```
# 기호와 함께 출력하기
+52
-52
 52
-52
```

- `{:+d}` 앞에 + 기호 추가하면 양수의 경우 + 붙여줌
- `{: d}`처럼 앞에 공백두면 양수의 경우 기호 위치를 공백으로 비워줌

➤ 예시 - 조합

```
01  # 조합하기
02  output_h = "{:+5d}".format(52)      # 기호를 뒤로 밀기: 양수
03  output_i = "{:+5d}".format(-52)     # 기호를 뒤로 밀기: 음수
04  output_j = "{:=+5d}".format(52)     # 기호를 앞으로 밀기: 양수
05  output_k = "{:=+5d}".format(-52)    # 기호를 앞으로 밀기: 음수
06  output_l = "{:05d}".format(52)      # 0으로 채우기: 양수
07  output_m = "{:05d}".format(-52)     # 0으로 채우기: 음수
08
09  print("# 조합하기")
10  print(output_h)
11  print(output_i)
12  print(output_j)
13  print(output_k)
14  print(output_l)
15  print(output_m)
```

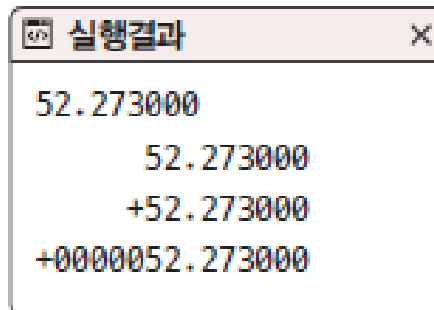
실행결과

```
# 조합하기
+52
-52
+ 52
- 52
+0052
-0052
```

● 부동 소수점 출력의 다양한 형태

➤ 예시 - float 자료형 기본

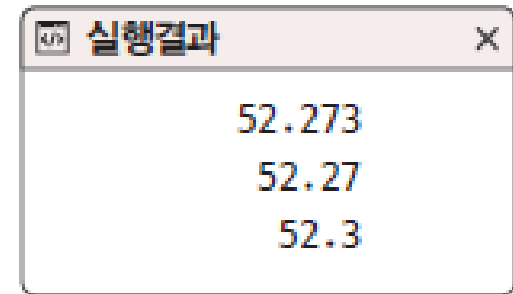
```
01 output_a = "{:f}".format(52.273)
02 output_b = "{:15f}".format(52.273)    # 15칸 만들기
03 output_c = "{:+15f}".format(52.273)    # 15칸에 부호 추가하기
04 output_d = "{:+015f}".format(52.273)    # 15칸에 부호 추가하고 0으로 채우기
05
06 print(output_a)
07 print(output_b)
08 print(output_c)
09 print(output_d)
```



```
실행결과
52.273000
52.273000
+52.273000
+0000052.273000
```

➤ 예시 - 소수점 아래 자릿수 지정하기

```
01 output_a="{:15.3f}".format(52.273)
02 output_b="{:15.2f}".format(52.273)
03 output_c="{:15.1f}".format(52.273)
04
05 print(output_a)
06 print(output_b)
07 print(output_c)
```

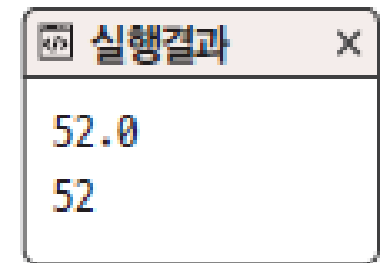


실행결과
52.273
52.27
52.3

- 의미 없는 소수점 제거하기

➤ 예시 - { :g }

```
01  output_a = 52.0
02  output_b = "{:g}".format(output_a)
03  print(output_a)
04  print(output_b)
```



실행결과

52.0
52

- upper() 함수

- 문자의 알파벳을 대문자로 바꿈

- lower() 함수

- 문자의 알파벳을 소문자로 바꿈

```
>>> a = "Hello Python Programming...!"  
>>> a.upper()  
'HELLO PYTHON PROGRAMMING...!'
```

```
>>> a.lower()  
'hello python programming...!'
```

- strip() 함수

- 문자열 양옆의 공백을 제거

- lstrip() 함수

- 왼쪽의 공백을 제거

- rstrip() 함수

- 오른쪽의 공백을 제거

➤ 의도하지 않은 줄바꿈 등의 제거

```
>>> input_a = """
    안녕하세요
문자열의 함수를 알아보니다
    """
```

```
>>> print(input_a)
```

```

    안녕하세요
문자열 함수를 알아보니다
```

```
>>> print(input_a.strip())
```

```
안녕하세요
문자열 함수를 알아보니다
```

- 문자열이 소문자로만, 알파벳으로만, 혹은 숫자로만 구성되어 있는지 확인
 - `isalnum()`: 문자열이 알파벳 또는 숫자로만 구성되어 있는지 확인합니다.
 - `isalpha()`: 문자열이 알파벳으로만 구성되어 있는지 확인합니다.
 - `isidentifier()`: 문자열이 식별자로 사용할 수 있는지 확인합니다.
 - `isdecimal()`: 문자열이 정수 형태인지 확인합니다.
 - `isdigit()`: 문자열이 숫자로 인식될 수 있는지 확인합니다.
 - `isspace()`: 문자열이 공백으로만 구성되어 있는지 확인합니다.
 - `islower()`: 문자열이 소문자로만 구성되어 있는지 확인합니다.
 - `isupper()`: 문자열이 대문자로만 구성되어 있는지 확인합니다.

- 불 (boolean)

- 출력이 True 혹은 False로 나오는 것

```
>>> print("TrainA10".isalnum())  
True  
>>> print("10".isdigit())  
True
```

● find()

- 왼쪽부터 찾아서 처음 등장하는 위치 찾기

● rfind()

- 오른쪽부터 찾아서 처음 등장하는 위치 찾기

```
>>> output_a = "안녕안녕하세요".find("안녕")
>>> print(output_a)
0
```

```
>>> output_b = "안녕안녕하세요".rfind("안녕")
>>> print(output_b)
2
```

● in 연산자

- 문자열 내부에 어떤 문자열이 있는지 확인할 때 사용
- 결과는 True(맞다), False(아니다)로 출력

```
>>> print("안녕" in "안녕하세요")  
True
```

```
>>> print("잘자" in "안녕하세요")  
False
```

- split() 함수

- 문자열을 특정한 문자로 자름

```
>>> a = "10 20 30 40 50".split(" ")
>>> print(a)
['10', '20', '30', '40', '50']
```

- 실행 결과는 리스트 (list)로 출력

- **format() 함수** : 숫자와 문자열을 다양한 형태로 출력
- **upper() 및 lower() 함수** : 문자열의 알파벳을 대문자 혹은 소문자로 변경
- **strip() 함수** : 문자열 양옆의 공백 제거
- **find() 함수** : 문자열 내부에 특정 문자가 어디에 위치하는지 찾을 때 사용
- **in 연산자** : 문자열 내부에 어떤 문자열이 있는지 확인할 때 사용
- **split() 함수** : 문자열을 특정한 문자로 자를 때 사용

- 함수와 그 기능을 연결해 보세요.

- | | | |
|-----------|---|------------------------|
| ① split() | • | • ㉠ 문자열을 소문자로 변환합니다. |
| ② upper() | • | • ㉡ 문자열을 대문자로 변환합니다. |
| ③ lower() | • | • ㉢ 문자열 양옆의 공백을 제거합니다. |
| ④ strip() | • | • ㉣ 문자열을 특정 문자로 자릅니다. |

- 다음 코드의 빈칸을 채워서 실행결과처럼 출력해 보세요.

```
a = input("> 1번째 숫자: ")
b = input("> 2번째 숫자: ")
print()

print("{} + {} = {}".format( , ))
```

