자료형

최석재
lingua@naver.com

여러 개의 값을 출력하기

- 여러 개의 값을 한 번에 출력하는 연습을 한다
- sep 아규먼트를 이용하면 다양한 기호를 구분자로 사용할 수 있다

```
name = "최길동"
age = 22
gender = "남자"

print(name, age, gender)                                    # 쉼표로 여러 내용을 연결
print()                                                     # 빈 줄 넣기
print("이름이", name, "인", age, "살", gender)               # 공백이 기본으로 들어간다
print("이름이", name, "인", age, "살", gender, sep="")       # 공백없이 출력
print("이름이", name, "인", age, "살", gender, sep=",")      # 공백대신 쉼표를 사용
```

정수형 int

- -2, -1, 0, 1, 2 와 같은 숫자를 정수라고 한다
- 변수에 정수를 넣으면 해당 변수는 '정수형'으로 저장된다

```
age = 22
print(age)
print(type(age))                # type() 함수를 이용하여 자료형을 확인할 수 있다

print()                         # 빈 줄 삽입

bigNumber = 2**124
print(bigNumber)                # 매우 큰 수도 잘 표현한다
print(type(bigNumber))
```

```
22
<class 'int'>

21267647932558653966460912964485513216
<class 'int'>
```

실수형 float

- -3.1, 3.25 등 소수점을 가진 형태를 실수라고 한다
- 실수형은 정수형보다 더 많은 저장공간을 차지한다

```
pi = 3.14
print(pi)
print(type(pi))          # 실수형으로 저장되었다

print()

pi = 3
print(pi)
print(type(pi))          # 정수형으로 변환되었다
```

```
3.14
<class 'float'>

3
<class 'int'>
```

부울형 bool

- 부울형은 참과 거짓을 판단하기 위하여 사용된다
- 값으로 True 또는 False를 입력하면 '부울형'으로 저장된다
- 부울형은 저장공간이 작으면서 논리적 흐름에 효과적으로 사용된다

```
found = True
print(found)
print(type(found))
print()

found = False
print(found)
print(type(found))
print()
```

```
True
<class 'bool'>

False
<class 'bool'>
```

- 부울값의 결과에 따라 특정 과정을 진행할지 여부를 결정할 수 있다

```
found = True

if(found):
    print("찾았습니다")
```

```python
print(20 > 13)                  # True
print(20 < 13)                  # False
print()

print('안녕' == '안녕')          # True
print('안녕' == '안녕 ')         # False
print()

number = 0
print(bool(number))             # False. 0은 False로 처리된다

number = 1
print(bool(number))             # True. 0 이외의 값은 True로 처리된다

number = 0.1
print(bool(number))             # True. 0 이외의 값은 True로 처리된다
```

문자열 연산 str

- 문자열은 text를 저장한다
- 문자열에는 연산 기호를 사용할 수 있다

```
mytext = "테스트 문자열입니다"
print(mytext)
print(type(mytext))                    # 문자열은 str로 표시된다

print("-" * 20)                        # 곱셈 기호로 문자열을 반복

print(mytext, mytext)                  # 앞에서와 같이 두 개의 문자열을 차례로 출력
print(mytext + mytext)                 # 덧셈 기호로 두 개의 문자열을 결합하여 출력
print(mytext + " " + mytext)           # 덧셈 기호로 세 개의 문자열을 결합하여 출력
```

특수 문자열

- 입력된 그대로가 아닌 특수한 기능을 갖는 문자열을 만들 때가 있다
- ₩n 은 줄을 띄우는 특수 문자열이다

```
mytext = "테스트 문자열입니다"
print(mytext)
print(mytext)                # print() 함수는 한 줄씩 띄운다

print("-" * 20)

print(mytext, "₩n")          # 기본 1줄에 1줄을 더 띄운다
print(mytext, "₩n₩n")        # 기본 1줄에 2줄을 더 띄운다
print(mytext)
```

```
테스트 문자열입니다
테스트 문자열입니다
--------------------
테스트 문자열입니다

테스트 문자열입니다


테스트 문자열입니다
```

여러 줄 문자열

- 여러 줄의 문자열도 간단하게 입력하고 출력할 수 있다

```
multitext = '''
여러 줄의 문자열을 출력할 수 있다
작은 따옴표 세 개 또는 큰 따옴표 세 개 """ 를
사용하여 여러 줄의 문자를 입력할 수 있다

그 내에서는 특수문자인 !@#$%^&* 도 자유롭게 표현할 수 있다
'''

print(multitext)
```

문자열 인덱싱

- 숫자와는 달리 문자열은 각 위치의 문자를 가져올 수 있다
- 이것을 문자열 인덱싱이라고 한다

```
mytext = "테스트 문자열입니다."
print(mytext[0])          # 첫 번째 문자
print(mytext[1])          # 1번은 두 번째 문자이다
print(mytext[3])          # 공백도 위치를 차지한다
print(mytext[4])
print(mytext[10])         # 마침표도 위치를 차지한다

mytext2 = 'abc'
print(mytext2[0])         # 영어도 한글과 같은 방식으로 처리된다
print(mytext2[-1])        # 역방향 접근도 가능하다. 가장 오른쪽이 -1이다
```

```
테
스
 
문
다
.
a
c
```

문자열 슬라이싱

- 한 글자가 아닌 여러 개의 문자열도 가져올 수 있다
- 이를 문자열 슬라이싱이라고 한다
- 복수의 문자이므로 시작 위치와 끝 위치를 지정해주어야 한다
- 끝 위치의 문자는 포함되지 않으니 주의해야 한다 (0부터 n개)

```
          0 1 2 3 4 5 6 7 8 9 10
mytext = "테스트 문자열입니다."
```

```
print(mytext[0:5])      # 콜론을 중심으로 시작 위치와 끝 위치를 지정한다
print(mytext[1:5])      # 하지만 끝 위치의 문자는 포함되지 않는다
print(mytext[1:])       # 끝 위치를 지정하지 않으면 마지막 문자까지 추출한다
print(mytext[:5])       # 시작 위치를 지정하지 않으면 처음부터 추출한다
print(mytext[:])        # 전체 출력
```

```
테스트 문
스트 문
스트 문자열입니다.
테스트 문
테스트 문자열입니다.
```

문자열은 불변

- 문자열은 인덱싱, 슬라이싱이 가능하지만 부분 변경은 되지 않는다
- 변경하려면 전체 값을 모두 바꿔야 한다

```
mytext = "테스트 문자열입니다."

print(mytext[1])
# mytext[1] = "츠"                # 에러 발생. 문자열의 부분을 수정할 수 없다

mytext = "테츠트 문자열입니다"
print(mytext)
```

문자열 메서드

- 문자열과 관련된 주요 메서드 몇 가지를 살펴본다

```
mytext = "Hello, my little dog!"

print(mytext.upper())          # 대문자로 변환
print(mytext)                  # 그러나 원 텍스트는 변하지 않는다 (문자열 불변성)
print(mytext.lower())          # 소문자로 변환
print(mytext.title())          # 각 단어의 첫 문자만 대문자로 변환
```

```
HELLO, MY LITTLE DOG
Hello, my little dog
hello, my little dog
Hello, My Little Dog
```

```python
print(mytext.count('t'))                # 특정 문자의 수를 세어준다
print(mytext.startswith('H'))           # 특정 문자로 시작하였는지를 판정한다
print(mytext.endswith('G'))             # 대소문자는 구분한다
print(mytext.upper().endswith('G'))     # 함수는 연결하여 사용할 수 있다

print("-" * 10)

mytext_list = mytext.split()            # 공백을 기준으로 문자열을 분리한다
print(mytext_list)                      # 분리된 결과는 리스트 타입으로 저장된다

mytext_list2 = mytext.split(',' )       # 특정 문자를 기준으로 분리할 수도 있다
print(mytext_list2)
```

```
2
True
False
True
----------
['Hello,', 'my', 'little', 'dog']
['Hello', ' my little dog']
```

리스트 list

- 리스트는 파이썬에서 가장 많이 사용하는 유형이다
- 여러 개의 변수를 하나로 묶은 형태이다
- 대괄호([ ])를 이용해서 만들며, 다양한 자료형을 담을 수 있다

```
count = [1, 2, 3, 4, 5]
cars = ['버스', '트럭', '승용차', '밴']
money = [1000, '1 dollar', '2달러', 3000]

print(count)
print(cars)
print(money)
```

```
[1, 2, 3, 4, 5]
['버스', '트럭', '승용차', '밴']
[1000, '1 dollar', '2달러', 3000]
```

```python
print(count[0])                    # 리스트 인덱싱
print(cars[1])
print(money[2])

all = [count, cars, money]         # 리스트를 담은 리스트
print(all)

a, b, c = all                      # 리스트의 각 원소를 분리할 수 있다
print(a, b, c)
```

```
1
트럭
2달러
[[1, 2, 3, 4, 5], ['버스', '트럭', '승용차', '밴'], [1000, '1 dollar', '2달러', 3000]]
[1, 2, 3, 4, 5] ['버스', '트럭', '승용차', '밴'] [1000, '1 dollar', '2달러', 3000]
```

리스트는 가변

```
수정 전: ['체다', '모짜렐라', '까망베르', '리코타']
수정 후: ['체다', '크림', '까망베르', '리코타']
['체다', '크림', '까망베르', '리코타', '고르곤']
['체다', '크림', '까망베르', '리코타']
```

- 리스트는 원소를 변경할 수 있다

```
cheeses = ['체다', '모짜렐라', '까망베르', '리코타']
print('수정 전:', cheeses)

cheeses[1] = '크림'                    # 1번 원소를 변경한다
print('수정 후:', cheeses)

cheeses2 = cheeses + ['고르곤']
print(cheeses2)                        # 새로운 리스트 생성
print(cheeses)                         # 원본 리스트는 변하지 않는다
```

리스트 슬라이싱

```
['체다', '모짜렐라', '까망베르', '리코타']
['모짜렐라', '까망베르']
['모짜렐라', '까망베르', '리코타']
['체다', '모짜렐라', '까망베르']
['체다', '모짜렐라', '까망베르', '리코타']
```

- 리스트는 인덱싱은 물론, 슬라이싱이 가능하다

```
cheeses = ['체다', '모짜렐라', '까망베르', '리코타']

print(cheeses)          # 전체 출력
print(cheeses[1:3])     # 1, 2 를 출력
print(cheeses[1:])      # 1번 이하를 모두 출력
print(cheeses[:3])      # 0, 1, 2 를 출력
print(cheeses[:])       # 전체 출력
```

리스트 메서드 1

```
['체다', '모짜렐라', '까망베르', '리코타']
1
['체다', '모짜렐라', '까망베르', '리코타', '고르곤']
['체다', '파마산', '모짜렐라', '까망베르', '리코타', '고르곤']
['체다', '파마산', '모짜렐라', '까망베르', '리코타', '고르곤', '콜비', '브리']
['체다', '파마산', '모짜렐라', '까망베르', '리코타', '고르곤', '콜비', '브리', '그리에르', '고다']
```

```
cheeses = ['체다', '모짜렐라', '까망베르', '리코타']

print(cheeses)
print(cheeses.index('모짜렐라'))     # 모짜렐라의 위치 1을 출력

cheeses.append('고르곤')             # 리스트에 원소 추가
print(cheeses)

cheeses.insert(1, '파마산')          # 1번 위치에 원소 추가
print(cheeses)

cheeses.extend(['콜비', '브리'])     # 둘 이상은 리스트로 구성
print(cheeses)

cheeses2 = ['그리에르', '고다']
cheeses3 = cheeses + cheeses2        # 리스트 결합으로도 가능하다
print(cheeses3)
```

리스트 메서드 2

```
정렬 전: ['체다', '모짜렐라', '까망베르', '리코타']
['까망베르', '리코타', '모짜렐라', '체다']
['체다', '모짜렐라', '리코타', '까망베르']
까망베르
['체다', '모짜렐라', '리코타']
['체다', '리코타']
1
```

```
cheeses = ['체다', '모짜렐라', '까망베르', '리코타']
print('정렬 전:', cheeses)

cheeses.sort()                    # 정렬 (원본 변경)
print(cheeses)

cheeses.reverse()                 # 역순 정렬
print(cheeses)

print(cheeses.pop())              # 마지막 원소를 가져온 뒤 삭제
print(cheeses)

cheeses.remove('모짜렐라')         # 특정 원소 삭제
print(cheeses)

print(cheeses.count('체다' ))      # 특정 원소 개수
```

파이썬 함수와 리스트

```
4
2000
0.2
2014.3400000000001
[0.2, 3.14, 11, 2000]
[3.14, 2000, 0.2, 11]
```

- 리스트와 함께 자주 사용되는 파이썬 함수로는 다음과 같은 것들이 있다

```
number = [3.14, 2000, 0.2, 11]

print(len(number))        # 원소의 개수
print(max(number))        # 가장 큰 수
print(min(number))        # 가장 작은 수
print(sum(number))        # 원소의 총합

print(sorted(number))     # 원소 정렬
print(number)             # sorted() 함수는 원본 리스트를 변경하지 않는다
```

튜플 tuple

```
(1, 2, 3)
(1, 'python', 3.14)
(1, (1, 2, 3), [4, 5, 6])
1 (1, 2, 3) [4, 5, 6]
```

- 리스트와 매우 비슷한 것으로 튜플이 있다
- 튜플이 리스트와 다른 점은 읽기 전용이어서 변경이 불가하다는 점이다
- 대신 속도가 빠르다

```
myTuple = (1, 2, 3)                    # 튜플은 괄호를 이용하여 만든다
print(myTuple)

myTuple = (1, "python", 3.14)          # 리스트와 같이 다양한 자료형을 저장할 수 있다
print(myTuple)

myTuple = (1, (1, 2, 3), [4, 5, 6]) # 튜플 자신은 물론, 리스트도 가능하다
print(myTuple)

a, b, c = myTuple                      # 튜플의 각 원소를 분리할 수 있다
print(a, b, c)
```

튜플 주의점

```
(1, 2, 3)
2
5000
<class 'int'>
(5000,)
<class 'tuple'>
```

```
myTuple = (1, 2, 3)
print(myTuple)
print(myTuple[1])
# myTuple[1] = 5          # 튜플의 원소는 변경할 수 없다

myTuple = (5000)          # 튜플 원소가 아닌, 튜플 변수의 변경은 가능하다

print(myTuple)
print(type(myTuple))      # 튜플이 아닌, 정수형이다!

myTuple = (5000,)         # 튜플의 원소가 하나만 있을 때는 쉼표로 끝낸다
print(myTuple)
print(type(myTuple))      # 튜플형
```

튜플의 인덱싱, 슬라이싱

```
('치즈', (1, 2, 3), [4, 5, 6], 3000)
치즈
((1, 2, 3), [4, 5, 6])
즈
2
```

- 튜플도 인덱싱과 슬라이싱이 가능하다

```
myTuple = ('치즈', (1, 2, 3), [4, 5, 6], 3000)
print(myTuple)
print(myTuple[0])              # 튜플 인덱싱
print(myTuple[1:3])            # 튜플 슬라이싱

# 원소가 원소를 가진 경우 다음과 같이 접근할 수 있다
# 리스트에서도 가능하다
print(myTuple[0][1])           # '즈' 출력
print(myTuple[1][1])           # 2 출력
```

튜플 메서드

```
(1, 2, 3, 4, 5, 6)
(1, 2, 3, 1, 2, 3, 1, 2, 3)
True
False
3
1
```

- 튜플은 항목 변경이 불가하므로, 리스트가 갖는 메서드의 일부만 갖는다

```
my_tuple = (1, 2, 3)
your_tuple = (4, 5, 6)

# 튜플 연산
our_tuple = my_tuple + your_tuple
print(our_tuple)                    # 리스트와 같이 + 기호로 결합할 수 있다

# 튜플 반복
multi_tuple = my_tuple * 3
print(multi_tuple)                  # 리스트와 같이 * 기호로 반복할 수 있다

# 멤버십 테스트
print(1 in my_tuple)                # True
print(3 not in my_tuple)            # False

# 찾기 관련 메서드
print(multi_tuple.count(1))         # 3. 몇 개 있는지
print(multi_tuple.index(2))         # 1. 몇 번째 있는지
```

튜플 관련 함수

```
4
2000
0.2
2014.3400000000001
[0.2, 3.14, 11, 2000]
(3.14, 2000, 0.2, 11)
```

- 리스트 관련 함수가 대부분 사용된다

```
number = (3.14, 2000, 0.2, 11)

print(len(number))      # 원소의 개수
print(max(number))      # 가장 큰 수
print(min(number))      # 가장 작은 수
print(sum(number))      # 원소의 총합

print(sorted(number))   # 원소 정렬. 결과는 리스트
print(number)           # sorted() 함수는 원본 리스트를 변경하지 않는다
```

집합 set

```
{1.0, 'hello', (1, 2, 3)}
<class 'set'>
(1, 2, [4, 5])
```

- 집합은 수학에서 배웠던 집합과 같은 개념이다
- 파이썬의 집합은 중복이 없고, 순서가 없다

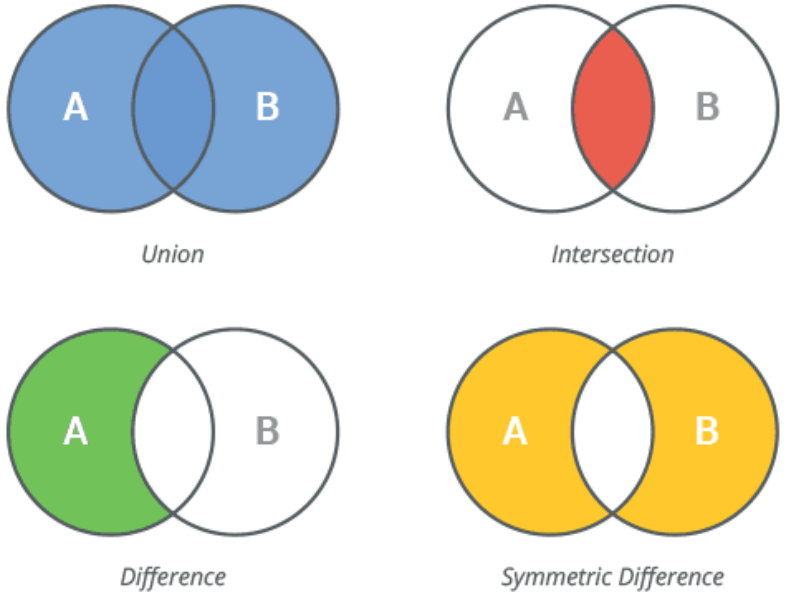
```python
my_set = {"hello", 1.0, (1, 2, 3)}
print(my_set)
print(type(my_set))          # set

# 집합은 중복을 허용하지 않으므로 가변객체를 가질 수 없게 하였다
# my_set2 = {1, 2, [4, 5]}   # 집합은 리스트를 가질 수 없다
my_set2 = (1, 2, [4, 5])     # 튜플은 리스트를 가질 수 있다
print(my_set2)

# print(my_set[0])           # 집합은 순서를 가지지 않으므로 인덱싱을 할 수 없다
# print(my_set[1:2])         # 집합은 순서를 가지지 않으므로 슬라이싱도 할 수 없다
```

집합 메서드

```
add: {'sky', 'sea', 'earth'}
update: {1, 2, 3, 'sea', 4, 'sky', 'earth'}
discard: None
{1, 2, 3, 4, 'sky', 'earth'}
pop: 1
{2, 3, 4, 'sky', 'earth'}
```

```
nature = {'sky', 'sea'}
print(nature)

nature.add('earth')                          # 항목 추가하기
print("add:", nature)                        # 추가되는 위치는 랜덤

nature.update({1, 2}, [2, 3], (3, 4))        # 여러 항목 추가는 update를 사용한다
print("update:", nature)                     # 다양한 자료형이 동일하게 저장, 중복은 삭제

print("discard:", nature.discard('sea'))     # 특정 요소의 삭제. 추출은 하지 않는다
print(nature)

print("pop:", nature.pop())                  # 랜덤 요소가 추출된 뒤 삭제된다
print(nature)
```

집합 연산




```
A = {1, 2, 3, 4, 5, 6}
B = {1, 2, 3, 7, 8, 9}
print("A:", A)
print("B:", B)

print("A | B:", A | B)          # 합집합
print("A & B:", A & B)          # 교집합
print("A - B:", A - B)          # 차집합
print("A ^ B:", A ^ B)          # 대칭차집합

print("부분집합:", {4, 5, 6} <= A)
print("부분집합:", {4, 5, 6} >= A)
```

```
A: {1, 2, 3, 4, 5, 6}
B: {1, 2, 3, 7, 8, 9}
A | B: {1, 2, 3, 4, 5, 6, 7, 8, 9}
A & B: {1, 2, 3}
A - B: {4, 5, 6}
A ^ B: {4, 5, 6, 7, 8, 9}
부분집합: True
부분집합: False
```

집합 관련 함수

```
True
False
4
2000
0.2
2014.3400000000001
[0.2, 3.14, 11, 2000]
{2000, 11, 3.14, 0.2}
```

```
number = {3.14, 2000, 0.2, 11}

# 멤버십 테스트
print(0.2 in number)
print(0.2 not in number)

print(len(number))      # 원소의 개수
print(max(number))      # 가장 큰 수
print(min(number))      # 가장 작은 수
print(sum(number))      # 원소의 총합

print(sorted(number))   # 원소 정렬. 결과는 리스트
print(number)           # sorted() 함수는 원본 리스트를 변경하지 않는다
```

사전 dict

```
{'name': '김영미', 'age': 25, 'job': 'singer'}
<class 'dict'>
{1: '색종이', 2: '포스터', 3: '플래카드'}
{1: '서울', 2: '부산', 3: '대전'}
```

- 사전은 키와 값의 쌍으로 이루어진 자료형이다

```
my_dict = {"name":"김영미", "age":25, "job":"singer"}        # 중괄호를 이용한다
print(my_dict)
print(type(my_dict))                                         # dict

my_dict = {1:"색종이", 2:"풍선", 2:"포스터", 3:"플래카드"}    # 키로 숫자를 이용
print(my_dict)                                               # 키는 중복될 수 없다

my_tuple = ([1, '서울'], [2, '부산'], [3, '대전'])           # 원소가 두 개인 튜플
my_dict = dict(my_tuple)                                     # dict() 함수 이용 가능
print(my_dict)
```

사전 사용하기

```
{'name': '김영미', 'age': 25, 'job': 'singer'}
김영미
김영미
None
{'name': '박희영', 'age': 25, 'job': 'singer', 'name2': '김재원'}
```

```
my_dict = {"name":"김영미", "age":25, "job":"singer"}
print(my_dict)

print(my_dict["name"])
print(my_dict.get("name"))      # 같은 결과

# print(my_dict["name2"])       # 에러 발생
print(my_dict.get("name2"))     # None 출력

my_dict["name"] = "박희영"       # 수정
my_dict["name2"] = "김재원"      # 새로운 값
print(my_dict)
```

사전 관련 메서드

```
items: dict_items([('name', '김영미'), ('age', 25), ('job', 'singer')])
keys: dict_keys(['name', 'age', 'job'])
values: dict_values(['김영미', 25, 'singer'])
age
age
김영미
{'age': 25, 'job': 'singer'}
```

```
my_dict = {"name":"김영미", "age":25, "job":"singer"}
print("items:", my_dict.items())          # item 확인
print("keys:", my_dict.keys())            # kye만 확인
print("values:", my_dict.values())        # value만 확인

x = list(my_dict.keys())                  # 결과를 리스트로 변환
print(x[1])

x = tuple(my_dict.keys())                 # 결과를 튜플로 변환
print(x[1])

print(my_dict.pop("name"))                # 값을 가져오면서 삭제하기
print(my_dict)
```

형 변환

```
7 <class 'str'>
7 <class 'int'>
7.0 <class 'float'>
```

- 자료형을 변환할 수 있다

```
### 작은 유형 사이의 변환 ###
strNum = '7'
print(strNum, type(strNum))          # str

intNum = int(strNum)
print(intNum, type(intNum))          # int

floatNum = float(strNum)
print(floatNum, type(floatNum))      # float
```

```
[1, 2, 3] <class 'list'>
(1, 2, 3) <class 'tuple'>
{1, 2, 3} <class 'set'>
```

```python
### 큰 유형 사이의 형변환 ###
my_list = [1, 2, 3]
print(my_list, type(my_list))        # list

my_tuple = tuple(my_list)
print(my_tuple, type(my_tuple))      # tuple

my_set = set(my_list)
print(my_set, type(my_set))          # set

# my_dict = dict(my_list)            # 현재 형태는 사전형으로 변경 불가
```

변수 출력하기 1

- 변수를 문장의 원하는 위치에 출력하는 방법을 다룬다
- 먼저 % 기호를 이용한 방법을 다룬다

- 변수가 놓일 자리, %, 출력될 내용의 세 부분이다
- 문장 안에 기호를 그대로 사용한다
- 출력될 내용은 튜플로 구성한다

```
print('%s' % ('하나'))                    # 1개 변수

print('%s, %s' % ('하나', '둘'))          # 2개 변수
```

```
하나
하나, 둘
```

변수출력 연습1

```
오늘 소개할 작품은 고흐의 1889년 작품 자화상입니다
물건이 1개 있어요.
물건이 10개 있어요.
물건이 100개 있어요.
물건이   1개 있어요.
물건이  10개 있어요.
물건이 100개 있어요.
원주율은 3.1입니다.
원주율은 3.14입니다.
원주율은  3.14입니다.
```

```
print("오늘 소개할 작품은 %s의 %d년 작품 %s입니다" % ("고흐", 1889, "자화상"))

print("물건이 %d개 있어요." % (1))
print("물건이 %d개 있어요." % (10))
print("물건이 %d개 있어요." % (100))

# 자릿수 확보
print("물건이 %3d개 있어요." % (1))          # 3개 숫자가 들어갈 공간을 확보
print("물건이 %3d개 있어요." % (10))
print("물건이 %3d개 있어요." % (100))

# 소수점 처리
print("원주율은 %.1f입니다." % (3.141592))    # 소수점 1자리까지
print("원주율은 %.2f입니다." % (3.141592))    # 소수점 2자리까지
print("원주율은 %5.2f입니다." % (3.141592))   # 5자리 공간 확보하고, 소수점 2자리까지
```

변수출력 연습2

```
체질량 지수 BMI를 계산합니다~

당신의 이름을 입력하세요: 김철수
몸무게를 입력하세요: 70
키를 입력하세요: 170
<class 'str'> <class 'str'>
<class 'float'> <class 'float'>

김철수님의 bmi는 70/(1.70*1.70)이므로 24.221입니다.
```

```
print("체질량 지수 BMI를 계산합니다~\n")

name = input("당신의 이름을 입력하세요: ")          # input() 함수로 입력을 받는다.
weight = input("몸무게를 입력하세요: ")             # 결과는 항상 문자열이다
height = input("키를 입력하세요: ")
print(type(weight), type(height))                  # 둘 다 str 타입이다

weight = float(weight)
height = float(height)
print(type(weight), type(height))                  # float로 바뀌었다

height = height / 100                              # bmi 계산을 위해 단위 조정
bmi = weight / (height*height)                     # 또는 weight / (height**2)

comment = "\n%s님의 bmi는 %d/(%.2f*%.2f)이므로 %.3f입니다."   # %f는 %d로 변환
print(comment % (name, weight, height, height, bmi))
```

```
하나
하나, 둘
하나, 둘

당신의 이름은?: 김철수
당신의 키는?(cm): 172
당신의 몸무게는?(kg): 72.1
```

변수 출력하기 2

- format() 메서드를 이용하는 방법은 보다 간편하다

```
print('{0}'.format('하나'))                  # 1개 변수
print('{0}, {1}'.format('하나', '둘'))       # 2개 변수
print('{}, {}'.format('하나', '둘'))         # 숫자를 써주지 않아도 된다

print()
name = input('당신의 이름은?: ')
height = float(input('당신의 키는?(cm): '))
weight = float(input('당신의 몸무게는?(kg): '))

height2 = height / 100                       # bmi 계산을 위해 단위 조정
bmi = weight / (height2*height2)             # 또는 weight / (height**2)
```

```
당신의 이름은?: 김철수
당신의 키는?(cm): 172
당신의 몸무게는?(kg): 72.1

김철수님의 키와 몸무게는 아래와 같습니다.
  172.0cm
   72.1kg

김철수님의 bmi는 72.1/(1.72*1.72)이므로 24.37입니다.

김철수님의 키, 몸무게, bmi는 아래와 같습니다.
 172.00cm
  72.10kg
  24.37
```

```python
# 자릿수의 공간을 확보할 때는 순서를 밝혀주면서 콜론을 사용해야 한다
print("\n{}님의 키와 몸무게는 아래와 같습니다.".format(name))
print("{0:7}cm".format(height))
print("{0:7}kg".format(weight))


# 소수점 정리를 할 때는 순서를 밝혀주면서 콜론을 사용해야 한다
comment = "\n{0}님의 bmi는 {1}/({2}*{2})이므로 {3:.2f}입니다."
print(comment.format(name, weight, height2, bmi))


# 공간 확보와 소수점 정리
print("\n{}님의 키, 몸무게, bmi는 아래와 같습니다.".format(name))
print("{0:7.2f}cm".format(height))
print("{0:7.2f}kg".format(weight))
print("{0:7.2f}".format(bmi))
```

변수 출력하기 3

- 출력할 내용을 변수로 만들어 재사용할 수 있다

```
# 변수 설정
basemat = "빵"

# 내용을 직접 쓰지 않고, 변수의 이름을 쓸 수 있다
print("{}은 좋은 밀가루를 써야 한다.".format(basemat))

# 사용할 변수가 둘 이상인 경우
print("{food}의 기본 재료는 {base}이다.".format(food='햄버거', base=basemat))
```