문자열형(String Type)

불변자료형(immutable)이다.

- 따라서 내용을 변경할 수도 없고, 문자의 순서를 바꿀 수도 없다.

시작과 끝을 따옴표로 묶어서 만든 순서가 있는 유니코드 글자(부호)의 배열이다.

- **시퀀스형**(sequence type) 자료 구조 중 하나다.
- '순서형'이라고도 부른다.

문자열 리터럴을 생성하기 위해서는 작은 따옴표 (' ... ') 또는 큰 따옴표 (" ... ")를 사용한다.

문자열 예시

In [ ]:
```
'안녕 파이썬'
```

In [ ]:
```
"안녕 파이썬"
```

In [ ]:
```
print('안녕 파이썬')
```

In [ ]:
```
print("안녕 파이썬")
```

In [ ]:
```
'안녕 파이썬"
```

작은따옴표로 시작했으면 작은따옴표로 끝나야 하고, 큰따옴표로 시작했으면 큰따옴표로 끝나야 한다.

만약 작은따옴표로 시작해서 큰따옴표로 끝나면(또는 그 반대) 오류가 난다.

In [ ]:
```
안녕 파이썬
```

따옴표를 생략해도 오류가 난다.

3~4장에서 설명했듯이 따옴표로 묶지 않은 문자는 **변수 이름**으로 인식한다.

중첩 따옴표

중첩 따옴표란 따옴표 안에 따옴표를 표현하는 것이다.

In [ ]:
```
print('문자열 안에 '작은 따옴표'를 사용하려면 어떻게 하나요?')
```

작은 따옴표가 들어 있는 경우는 큰 따옴표로 문자열을 생성하면 된다.

In [ ]:
```
print("문자열 안에 '작은 따옴표'를 사용하려면 큰 따옴표로 묶는다.")
```

큰 따옴표가 들어 있는 경우는 작은 따옴표로 문자열을 생성하면 된다.

In [ ]:
```
print('문자열 안에 "큰 따옴표"를 사용하려면 작은 따옴표로 묶는다.')
```

이스케이프 문자인 역슬래시(\)를 사용하면 따옴표를 문자열 리터럴을 생성하는 특수문자가 아닌 일반 문자열로 표현하는 것이 가능하다.

[참고]

- 윈도우용 한글 키보드는 '₩'가 '\'와 같은 기능을 한다.

In [ ]:
```
print('문자열 안에 같은 \'따옴표\'를 사용하려면 이스케이프 문자를 사용한다.')
```

In [ ]:
```
print("문자열 안에 같은 \"따옴표\"를 사용하려면 이스케이프 문자를 사용한다.")
```

삼중 따옴표

삼중 따옴표란 따옴표를 3개(triple quotes) 사용하는 것이다.

- 작은 따옴표 3개(''' ... ''') 또는 큰 따옴표 3개(""" ... """)를 사용한다.

삼중 따옴표 예시

In [ ]:
```
'''안녕 파이썬'''
```

In [ ]:
```
"""안녕 파이썬"""
```

In [ ]:
```
print('''안녕 파이썬''')
```

In [ ]:
```
print("""안녕 파이썬""")
```

따옴표 안에 따옴표를 넣을 때 일일이 역슬래시(\)를 넣어주지 않아도 된다.

In [ ]:
```
print('''문자열 안에 '작은 따옴표'나 "큰 따옴표"를 사용할 수 있다.''')
```

문자열 리터럴을 여러 줄에 나눠 입력할 때 편리하다.

In [ ]:
```
print('''문자열 안에 '작은 따옴표'나
"큰 따옴표"를 사용할 수도 있고
여러 줄을 사용할 때도 편리하다.''')
```

이스케이프 문자

- 탈출(escape) 문자라고도 하며 역슬래시(\ 또는 ₩)로 시작한다.
- 이스케이프 문자인 역슬래시(\)는 역슬래시 바로 다음에 따라오는 문자를 원래 의미가 아닌 특별한 의미로 처리해야 할 때 사용하는 문자다.

\n : 문자 'n'으로 처리하지 않고 '새줄바꿈'(newline)으로 처리한다.

- 새줄바꿈 문자 : 한 줄의 끝남을 표시하는 문자로 '개행 문자' 또는 'EOL(end-of-line)'과 같은 뜻이다.

In [ ]:
```
print('첫 번째 줄\n두 번째 줄\n세 번째 줄')
```

\t : 문자 't'로 처리하지 않고 '탭(tab)' 문자로 처리한다.

In [ ]:

```
print('첫 번째 칸\t두 번째 칸\t세 번째 칸')
```

\' : 작은 따옴표를 파이썬 문자열 리터럴을 생성하는 특수 문자로 사용하지 않고 일반적으로 알려진 작은 따옴표 문자로 처리한다.

In [ ]:

```
print('작은 따옴표 안에 \'작은 따옴표\'를 사용할 수 있다.')
```

\" : 큰 따옴표를 파이썬 문자열 리터럴을 생성하는 특수 문자로 사용하지 않고 일반적으로 알려진 큰 따옴표 문자로 처리한다.

In [ ]:

```
print("큰 따옴표 안에 \"큰 따옴표\"를 사용할 수 있다.")
```

퀴즈

print() 함수를 한번만 사용해서 다음 문장을 3가지 방법으로 출력하시오.

```
President Barack Obama said,
 "Don’t just play on your phone,
    program it."
```

[참고]

- 두 번째 줄 첫 시작에 빈 공백 한 개를 삽입한다.
- 세 번째 줄 첫 시작에 탭 한 개 삽입한다.

1. 삼중 따옴표로 문자열 생성하시오.
2. 작은 따옴표로 문자열 생성하시오.
3. 큰 따옴표로 문자열 생성하시오.

In [ ]:

```
print("""President Barack Obama said,
 "Dont't just play on your phone,
    program it." """)
```

In [ ]:

```
print('President Barck Obama said,\n "Don\'t jus play on your phone\n\tprogram it."')
```

In [ ]:

```
print("President Barck Obama said,\n \"Don't jus play on your phone\n\tprogram it.\"")
```

로스트링

문자열 내에서 역슬래시(\)가 이스케이프 문자가 아니라 원래 의미인 역슬래시 문자라면?

In [ ]:

```
print('C:\home\nations')
```

역슬래시 앞에 이스케이프 문자를 넣어 역슬래시를 이스케이프 문자로 처리하지 말고 일반 문자인 '\'로 처리하면 된다.

In [ ]:

```
print('C:\home\\nations')
```

그런데 문자열 내에 역슬래시 문자를 많이 사용해야 한다면?

In [ ]:

```
print('C:\home\nations\name\nationality\nature')
```

In [ ]:

```
print('C:\home\\nations\\name\\nationality\\nature')
```

다수의 역슬래시를 사용해야 해서 코드가 매우 복잡해질 수 있다.

이런 경우 **로스트링**(raw string)을 사용하면 된다.

In [ ]:

```
print(r'C:\home\nations\name\nationality\nature')
```

로스트링(raw string)이란?

- 문자열을 나타내는 첫 따옴표 앞에 **r**을 붙여
- 해당 문자열 내부의 모든 문자는 있는 그대로 인식하게 한다.
- 따라서, 걱정없이 이스케이프 문자를 마음껏 사용할 수 있다.

화이트스페이스

화이트스페이스(whitespace)란?

화면상으로는 표시되지 않지만 컴퓨터 모니터 화면이나 프린터 등과 같은 출력 장치를 제어하는 문자를 말한다.

공백문자(' '), 새줄바꿈('\n'), 탭('\t'), 캐리지 리턴('\r'), 수직탭('\v' 또는 '\x0b'), 페이지 넘기기('\f' 또는 '\x0c') 등이 있다.

파이썬이 사용하는 화이트스페이스를 확인하려면...

In [ ]:

```
import string
string.whitespace
```

화이트스페이스 문자의 역할은 **print()** 함수를 사용해서 텍스트 형식으로 출력해보면 쉽게 알 수 있다.

In [ ]:

```
x = '\t일\n월\n화\n수\n목\n금\n\t토'
```

In [ ]:

```
x          # 대표 형식으로 출력한다.
```

In [ ]:

```
print(x)  # 텍스트 형식으로 출력한다.
```

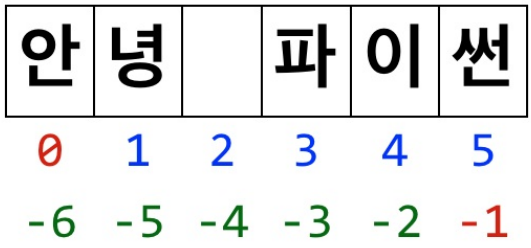
문자열 인덱스

문자열은 **시퀀스형**(sequence) 자료 구조다.

문자열은 각 **문자**의 위치가 정해져 있는 즉, 순서가 있는 문자의 배열이다.

- 문자는 말 그대로 한 개의 문자를 뜻한다
- 문자가 순서를 가지고 정해진 위치에 들어가게 되면 문자의 배열인 문자열을 생성하게 된다.

문자열 인덱스(index)란?



인덱스는 문자열 배열 안에서 각 값의 위치다.

- 첫 문자의 인덱스 : **[0]**

- 마지막 문자의 인덱스 : **[n - 1]** 또는 **[-1]**
  - ***n*** = 문자열의 전체 크기
  - 인덱스 **[-1]**을 사용하면 항상 마지막 문자를 찾을 수 있으므로 매우 유용하다.

IndexError 예외

- 문자열 범위를 벗어난 인덱스나 빈 문자열에 접근하려고 하면 IndexError 예외가 발생한다.

문자열 인덱스 예시

In [ ]:
```
s = '안녕 파이썬'
```

In [ ]:
```
s[0]   # '안녕 파이썬'
```

In [ ]:
```
s[2]   # '안녕 파이썬'
```

In [ ]:
```
s[6]   # '안녕 파이썬'
```

In [ ]:
```
s[-1]  # '안녕 파이썬'
```

In [ ]:
```
s[-3]  # '안녕 파이썬'
```

In [ ]:
```
s[-7]  # '안녕 파이썬'
```

문자 한 개 이상 여러 개를 한꺼번에 가져올 수는 없을까?

예를 들어, '안녕 파이썬'에서 '파이썬'을 한 번에 추출할 수는 없을까?

가능하다.

In [ ]:

```
s[3:]  # '안녕 파이썬'
```

인덱스는 문자를 한 개씩만 가져올 수 있지만,

문자열 **분할** 연산자를 사용하면 문자열의 일부 또는 전부를 추출할 수 있다.

문자열 분할

문자열 분할(slicing)이란?

분할 연산자 **[:]** 또는 **[: :]**를 사용하여 문자열의 일부 또는 전부를 추출하는 것을 말한다.

문자열[시작번호:끝번호]

문자열의 ***시작번호***부터 ***끝번호*** 바로 앞까지의 문자를 추출한다.

- ***끝번호***에 해당하는 문자는 **포함하지 않는다**.

시작번호와 ***끝번호***는 생략할 수 있다.

- ***시작번호***를 생략하면 기본 값인 **0**에서 시작한다.
- ***끝번호***를 생략하면 ***문자열***의 마지막 문자까지 추출한다.
  - 즉, 기본 값인 **len(문자열)**까지 추출하는 것과 같은 결과다.

따라서, ***시작번호***와 ***끝번호***를 모두 생략하면 대상 ***문자열*** 전체를 가져오게 된다.

- **문자열[:]**은 **문자열[0:len(문자열)]**과 같다.

In [ ]:

```
s = 'abcde 12345'  # len(s) = 11
```

In [ ]:

```
s[0:len(s)]        # 'abcde 12345'
```

In [ ]:

```
s[:]               # 'abcde 12345'
```

In [ ]:

```
s[2:-1]            # 'abcde 12345'
```

In [ ]:

```
s[2:]              # 'abcde 12345'
```

In [ ]:

```
s[-7:]             # 'abcde 12345'
```

In [ ]:

```
s[-1:]             # 'abcde 12345'
```

문자열[시작번호:끝번호:폭]

폭(step)은 설정한 숫자만큼씩 건너 뛰며 추출한다.

폭의 기본값은 1이다. 따라서, ***폭***을 생략하면 1이기 때문에 문자열 시작부터 끝의 바로 앞에 있는 모든 문자를 추출한다.

In [ ]:

```
s[:]            # 'abcde 12345'
```

In [ ]:

```
s[::]           # 'abcde 12345'
```

In [ ]:

```
s[::1]          # 'abcde 12345'
```

만약 폭(step)이 음의 정수라면?

폭이 음수라면 문자열의 시작과 끝이 바뀌어 뒤쪽 끝에서 출발해서 앞쪽으로 폭만큼 건너뛰면서 문자를 추출한다.

In [ ]:

```
s[::-1]          # 'abcde 12345'
```

In [ ]:

```
s[::2]      # 'abcde 12345'
```

In [ ]:

```
s[::-2]     # 'abcde 12345'
```

In [ ]:

```
s[:-4:3]    # 'abcde 12345'[0:-4:3]와 같다.
```

In [ ]:

```
s[:4:-2]    # 'abcde 12345'[-1:4:-2]와 같다.
```

In [ ]:

```
s[2:4:-2]   # 'abcde 12345'
```

In [ ]:

```
s[4:2:-2]
```

인덱스 접근

In [ ]:

```
s[11]    # 'abcde 12345'
```

범위를 벗어난 인덱스에 접근하면 예외가 발생한다.

하지만...

분할 연산자를 사용하면 범위를 벗어난 인덱스에 접근해도 오류가 나지 않는다.

In [ ]:

```
s[11:]   # 'abcde 12345'
```

In [ ]:

```
s[10:11]  # 'abcde 12345'
```

In [ ]:

```
s[6:99]  # 'abcde 12345'
```

문자열 수정

문자열은 **불변자료형**이기 때문에 문자열의 일부를 교체/추가/삭제 할 수 없다.

- 불변자료형(immutable) : 일단 값을 생성한 후에는 일부나 전부를 변경할 수 없는 자료형을 말한다. 따라서 수정이 불가능하다.

문자열의 일부분을 다른 문자열로 교체하거나 문자를 추가 또는 삭제하면 예외가 발생한다.

In [ ]:

```
s = 'Python'
s[1] = 'i'  # Pithon으로 변경
```

그러나...

앞서 실습에서 했듯이, 문자열 분할 연산자를 사용해서 필요한 문자만 추출한 후 **+** 연산자로 결합하면 된다.

그리고 기존 변수에 새로 결합한 문자열을 재할당하면

마치 기존 문자열을 새로운 문자열로 바꾼 것과 같은 효과를 볼 수 있다.

따라해보기

In [ ]:

```
s = '초급자를 위한 파이썬'
len(s)
```

In [ ]:

```
# '초급자를 위한 파이썬'
s[0] = '중'  # 첫 문자를 '중'으로 대체한다. => 예외 발생!!!
```

In [ ]:

```
# '초급자를 위한 파이썬'
s = '중' + s[1:]
print(s)
```

In [ ]:

```
# '중급자를 위한 파이썬'
s = s[:8] + '재미있는 ' + s[8:]
print(s)
```

자주 사용하는 문자열 메소드

- 문자열 분할 메소드
- 문자열 결합 메소드
- 문자열 삭제 메소드
- 문자열 교체 메소드

문자열 분할 메소드

***문자열*.split(*sep=None, maxsplit=-1*)**

***문자열*.rsplit(*sep=None, maxsplit=-1*)**

이 두 메소드는 문자열을 **리스트** 형태로 반환한다.

특징은 다음과 같다.

- **split()** 메소드는 ***sep***를 기준으로 문자열을 왼쪽부터 나눈 후 리스트 형태로 반환한다.
- **rsplit()** 메소드는 ***sep***를 기준으로 문자열을 오른쪽부터 나눈 후 리스트 형태로 반환한다.
- ***sep***는 **구분자**로 문자열을 분할할 때의 기준이다.
  - 구분자 ***sep***를 생략하면 기본값인 화이트스페이스 문자 기준으로 분할한다.
  - 구분자 ***sep***는 반환하는 리스트에 포함되지 않는다.
- ***maxsplit***는 최대 분할 값으로 이 값만큼만 구분자 ***sep***로 분할한다.
  - 만약 문자열에서 구분자 ***sep***의 개수가 최대 분할 값 ***maxsplit***보다 작다면 구분자 ***sep***를 기준으로 모두 분할한다.
  - 최대 분할 값 ***maxsplit***을 생략하거나 기본값인 **-1**이 주어지면 구 분자 ***sep***만큼 모두 분할한다.

In [ ]:

```
kor = '파이썬을 배우면서 파이썬을 즐기자!!!'
```

In [ ]:

```
# 기본값은 화이트스페이스 문자 기준으로 모두 분할해서 리스트로 반환한다.
kor.split()
```

In [ ]:

```
# --- '파이썬을 배우면서 파이썬을 즐기자!!!'
# '을' 기준으로 모두 분할한다.
'파이썬을 배우면서 파이썬을 즐기자!!!'.split('을')  # split(sep='을')와 같다.
```

In [ ]:

```
# --- '파이썬을 배우면서 파이썬을 즐기자!!!'
# '을' 기준으로 오른쪽부터 한 번만 분할한다.
kor.rsplit('을', 1)  # rsplit(sep='을', maxsplit=1)와 같다.
```

In [ ]:

```
# --- '파이썬을 배우면서 파이썬을 즐기자!!!'
# 화이트스페이스 문자 기준으로 왼쪽부터 두 번만 분할한다.
kor.split(maxsplit=2)
```

In [ ]:

```
# --- '파이썬을 배우면서 파이썬을 즐기자!!!'
# 화이트스페이스 문자 기준으로 오른쪽부터 한 번만 분할한다.
kor.rsplit(maxsplit=1)
```

In [ ]:

```
eng = 'Introduction to Python'
```

In [ ]:

```
# 화이트스페이스 문자 기준으로 오른쪽부터 모두 분할한다.
eng.rsplit()
```

In [ ]:

```
# --- 'Introduction to Python'
# 't' 기준으로 왼쪽부터 두 번만 분할한다.
eng.split(sep='t', maxsplit=2)
```

In [ ]:

```
# --- 'Introduction to Python'
# 'o'를 기준으로 오른쪽부터 한 번만 분할한다.
eng.rsplit('o', maxsplit=1)
```

In [ ]:

```
# --- 'Introduction to Python'
# 'o'를 기준으로 오른쪽부터 모두 분할한다.
eng.rsplit('o')  # eng.split('o')와 결과는 같다.
```

[주의]

지정한 구분자 ***sep***가 문자열의 맨 끝에 있으면 반환하는 리스트의 마지막에 빈 문자열을 포함한다.

split()는 구분자를 기준으로 양쪽을 반환하는데 구분자의 오른쪽에 아무 것도 없기 때문에 오른쪽 끝은 빈 문자열로 반환한다.

In [ ]:
```
'Introduction to Python'.split('n')
```

In [ ]:
```
'두 줄입니다\n'.split('\n')
```

[주의]

구분자 ***sep***를 지정하지 않았다면, 연속적으로 있는 화이트스페이스를 하나의 구분자로 취급한다.

따라서 빈 문자열이나 화이트스페이스로만 이루어져 있는 문자열은 빈 리스트를 반환한다.

In [ ]:
```
'  \t   \n  \n\t   \t  '.split()
```

In [ ]:
```
'                     '.split()
```

In [ ]:
```
''.split()
```

In [ ]:
```
'   1   2   3   '.split()
```

In [ ]:
```
'  \t 1  \n  2\n\t   \t3  '.split()
```

문자열 결합 메소드

***결합문자열*.join(*순회형*)**

- 여러 문자열을 한꺼번에 연결할 때 사용하며, ***순회형*** 자료를 ***결합문자열***로 결합해서 문자열 한 개로 반환하는 함수다.

특징은 다음과 같다.

- 여러 문자열을 한꺼번에 연결할 때 사용하며, ***순회형*** 자료를 ***결합문자열***로 결합해서 문자열 한 개로 반환하는 함수다.
- ***결합문자열***은 문자열을 결합할 때 사용하는 문자열이다.
- ***순회형***(iterable)은 순회 또는 반복이 가능한 자료형으로 문자열, 리스트, 튜플, 딕셔너리, 세트 등이 있다.
  - 순회형의 문자열들을 ***결합문자열***로 결합해서 하나의 문자열로 반환하기 때문에 순회형의 모든 객체들은 반드시 문자열이어야 한다.

In [ ]:
```
instruments = ['drum', 'guitar', 'bass', 'keyboard']
```

In [ ]:
```
# 리스트의 모든 문자열 객체를 공백문자로 결합한다.
' '.join(instruments)
```

In [ ]:
```
# --- instruments = ['drum', 'guitar', 'bass', 'keyboard']
# 리스트의 모든 문자열 객체를 쉼표와 공백문자로 결합한다.
', '.join(instruments)
```

In [ ]:

```
# --- instruments = ['drum', 'guitar', 'bass', 'keyboard']
# 리스트의 모든 문자열 객체를 '=><=' 결합문자열로 결합한다.
'=><='.join(instruments)
```

In [ ]:

```
# 리스트 안에 문자열이 아닌 숫자 정수가 들어있다.
'+'.join(['파이썬', 5])
```

문자열 삭제 메소드

***문자열*.strip(*[삭제할-문자-세트]*)**

***문자열*.lstrip(*[삭제할-문자-세트]*)**

***문자열*.rstrip(*[삭제할-문자-세트]*)**

이 세 메소드는 ***문자열***에서 ***삭제할-문자-세트***를 제거한 문자열을 반환하는 함수다.

특징은 다음과 같다.

- **strip()**은 ***삭제할-문자-세트***를 ***문자열***의 양쪽 끝부터 삭제한 문자열을 반환한다.
- **lstrip()**은 ***삭제할-문자-세트***를 ***문자열***의 왼쪽 끝부터 삭제한 문자열을 반환한다.
- **rstrip()**은 ***삭제할-문자-세트***를 ***문자열***의 오른쪽 끝부터 삭제한 문자열을 반환한다.
- ***삭제할-문자-세트***는 삭제할 문자들의 집합으로 ***문자열***의 끝부터 ***삭제할-문자-세트***에 포함되지 않은 문자에 도달할 때까지 계속 삭제한다.
- 이때 ***삭제할-문자-세트***에 포함된 모든 문자를 ***문자열***에서 삭제하기 때문에 ***삭제할-문자-세트***에 포함된 문자의 순서는 중요하지 않다.
- 만약 ***삭제할-문자-세트***를 지정하지 않으면 기본값인 화이트스페이스 문자를 ***문자열***에서 삭제한다.

In [ ]:

```
s = '\t 초급자를\n위한 파이썬!!!'
```

In [ ]:

```
# 문자열 양쪽 끝부터 화이트스페이스 문자를 삭제한다.
s.strip()
```

In [ ]:

```
# --- '\t 초급자를\n위한 파이썬!!!'
# 문자열 왼쪽 끝부터 화이트스페이스 문자를 삭제한다.
s.lstrip()
```

In [ ]:

```
# --- '\t 초급자를\n위한 파이썬!!!'
# 문자열 오른쪽 끝부터 화이트스페이스 문자를 삭제한다.
s.rstrip()
```

In [ ]:

```
# --- '\t 초급자를\n위한 파이썬!!!'
# 문자열 양쪽 끝부터 '\t'와 '!' 문자를 삭제한다.
s.strip('!\t')
```

In [ ]:

```
# --- '\t 초급자를\n위한 파이썬!!!'
# 문자열 양쪽 끝부터 '\t', '!', 공백문자를 삭제한다.
s.strip('! \t')
```

In [ ]:

```
# 문자열의 양쪽 끝부터 'goPniR'에 포함된 문자를 삭제한다.
'Right way to Python'.strip('goPniR')
```

문자열 교체 메소드

***문자열*.replace(*기존-부분문자열, 새로운-부분문자열[, 교체횟수]*)**

특징은 다음과 같다.

- ***문자열*** 중 ***기존-부분문자열***을 모두 ***새로운-부분문자열***로 교체한 새 문자열을 반환한다.
- 선택 사항인 ***교체횟수***(count)를 지정하면, 그 ***교체횟수***만큼만 교체한 새 문자열을 반환한다.
- ***부분문자열***이란 ***문자열***의 일부 또는 전부를 뜻한다.

In [ ]:

```
# 문자열에 '초'가 있으면 '중'으로 교체한다.
'초급자를 위한 파이썬'.replace('초', '중')
```

In [ ]:

```
# 소문자 'o'를 대문자 'O'로 두 번만 교체한다.
'I love Python programming'.replace('o', 'O', 2)
```

문자열 대문자/소문자 변환

- ***upper***: 모든 영문자를 대문자로 변환
- ***lower***: 모든 영문자를 소문자로 변환

In [ ]:

```
s = 'Right way to Python'
print(s.upper())
print(s.lower())
```

출력 간단히 하기

다음 코드는 출력 결과는 간단하지만 출력하는 방법은 복잡하다.

In [ ]:

```
x = 3
y = 5
z = 19

print(str(x) + ' * ' + str(y) + ' = ' + str(z) + ' (' + str(x * y == z) + ')')
```

하지만...

***문자열*.format()** 메소드를 이용하면 출력을 편하게 할 수 있다.

In [ ]:

```
print('{} * {} = {} ({})'.format(x, y, z, x * y == z))
```

다음에 배울 **for** 순환문에서 **format()**을 사용하면 출력을 간편하게 할 수 있다.

In [ ]:

```
for i in range(1, 10):
    print('{} * {} = {}'.format(i, 5, i * 5))
```

문자열 서식 : format() 메소드

***문자열*.format(**위치-전달인자, **키워드-전달인자*)**

특징은 다음과 같다.

- ***문자열***의 서식 설정 혹은 포맷과 관련된 작업을 실행한다.
- 대체 필드(replacement field)란 ***문자열*** 안의 특정 값을 바꿔야 할 때 사용한다.
  - **{ }**로 표시한다.
- 이 메소드를 호출하는 ***문자열***은 문자열 리터럴과 대체 필드를 포함할 수 있다. (예, ***'바로 쓰는 {}'***.format('파이썬'))
- 각 대체 필드를 **format()** 메소드의 전달인자로 대체한 새 문자열을 반환한다.
  - 전달인자(argument)란 **format()** 메소드의 소괄호 안으로 전달되는 값이나 변수를 말한다. (예, '바로 쓰는 {}'.format('**파이썬**'))

- 각 대체 필드에는 ***위치-전달인자***의 인덱스 번호나 ***키워드-전달인자***의 키워드 식별자가 들어 있다.

- ***위치-전달인자***를 사용한다면 대체 필드인 **{ }** 안에 전달인자의 위치를 알려주는 인덱스 번호(0, 1, ...)가 들어간다.
  - 단, 파이썬 버전 3.1 이후부터는 인덱스 번호를 생략할 수 있다.
- ***위치-전달인자***(positional argument)란 앞에 키워드 식별자가 없는 전달인자를 말한다.
  - 예를 들어 *문자열*.format(***'A4', 50, False***)같은 것이 이에 해당한다.
  - ***위치-전달인자***의 위치 인덱스 번호는 **0**부터 시작한다. 즉, 'A4'의 인덱스 번호는 **0**이며, 50의 인덱스 번호는 **1**, **False**의 인덱스 번호는 **2**다.

- ***키워드-전달인자***를 사용한다면 대체 필드인 **{ }** 안에 전달인자의 키워드 식별자를 입력한다.
- ***키워드-전달인자***(keyword argumnts)란 키워드 식별자가 오는 전달인자를 말한다.
  - 예를 들어 *문자열*.format(***paper='A4', copies=50, color=False***) 같은 것이 이에 해당한다.
  - ***paper***, ***copies***, ***color***가 키워드 식별자다.

따라해보기 : 위치 전달인자 사용해보기

In [ ]:
```
'1 + 2 = {0}'.format(1 + 2)
```

In [ ]:
```
'1 + 2 = {}'.format(1 + 2)
```

In [ ]:
```
'{0} * {1} = {2}입니다.'.format(3, 5, 3 * 5)
```

In [ ]:
```
'{} * {} = {}입니다.'.format(3, 5, 3 * 5)
```

In [ ]:
```
'{1} * {0} = {2}입니다.'.format(3, 5, 3 * 5)
```

In [ ]:
```
for i in range(1, 10):  # i의 값이 1부터 9까지 주어짐
    print('{0} * {1} = {2}'.format(5, i, 5 * i))
```

In [ ]:
```
for i in range(1, 10):  # i의 값이 1부터 9까지 주어짐
    print('{} * {} = {}'.format(5, i, 5 * i))
```

따라해보기 : 키워드 전달인자 사용해보기

키워드 전달인자를 사용하면 전달인자의 위치는 중요하지 않다.

즉, 전달인자의 위치와 무관하게 키워드 식별자를 통해 값을 전달하기 때문이다.

In [ ]:
```
for i in range(1, 10): # i값은 1부터 9까지다.
    print('{left} * {right} = {result}'.format(result=5*i, right=i, left=5))
```

따라해보기 : 자주 사용하는 format() 메소드의 숫자 서식 설정해보기

In [ ]:

```
# 쉼표로 정수를 그룹화해서 출력하기
'{0:,}'.format(123456789)
```

In [ ]:

```
# 실수를 소수점 두 번째 자리로 반올림해서 출력하기
'{:.2f}'.format(123456789.12534)
```

In [ ]:

```
# 실수를 소수점 세 번째 자리로 반올림해서 출력하기
'{:.3f}'.format(123456789.12534)
```

In [ ]:

```
# 쉼표로 실수를 그룹화하고 소수점 두 번째 자리로 반올림해서 출력하기
'{:,.2f}'.format(123456789.12534)
```

f-string (Literal String Interpolation)

- python 3.6 이상부터 지원

In [ ]:

```
for i in range(1,10):
    print(f'{i} * {5} = {i*5}')
```