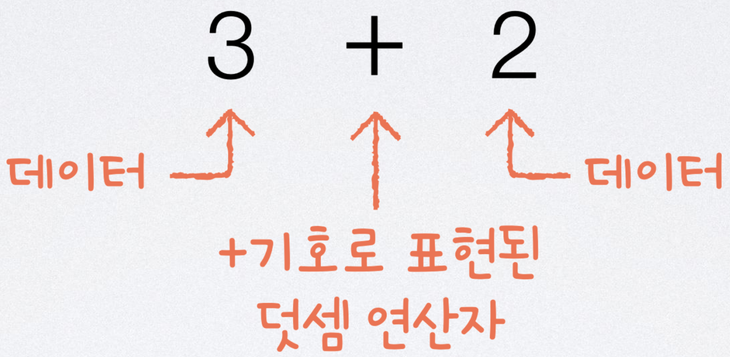
02-2 문자 데이터 처리

문자 데이터 **연결** 연산자 +

연산자란 어떤 기능을 하는 **명령어**를 **기호**로 표시한 것



연산자는 항상 데이터와 함께 쓰임

문자 데이터 **연결 연산자(+)**는

서로 다른 문자 데이터를 연결해서 **하나의 새로운 문자 데이터**를 만듦

직접 해보는 손코딩 

소스 코드 str02.py

```
01  print("2020년을" + " " + "빛낼" + " " + "프로그래밍" + " " + "입문서!")
02  print("혼자" + " " + "공부하는" + " " + "첫" + " " + "프로그래밍!")
03  print("혼공" + "족")
04  print("혼공" + "프로")
05  print("혼공" + "파")
```

문자 데이타 **반복 연결** 연산자 *

문자 데이터 **반복 연결 연산자(*)**는

문자 데이터를 n회 반복 연결해서 **하나의 새로운 문자 데이터**를 만듦

반복 연결할 데이터 * 반복 횟수

문자 데이터 **반복 연결 연산자(*)**는

문자 데이터를 n회 반복 연결해서 **하나의 새로운 문자 데이터**를 만듬

```
"=" * 5
```

```
"=====" 
```

직접 해보는 손코딩 

소스 코드 str03.py

```
01  print("*" * 2)
02  print(" " * 2 + "*")
03  print(" " * 3 + 1 * "*")
04  print(" " * 4 + 1 * "*")
05  print(" " * 5 + 2 * "*")
```

문자 데이터 **길이** 알아내기

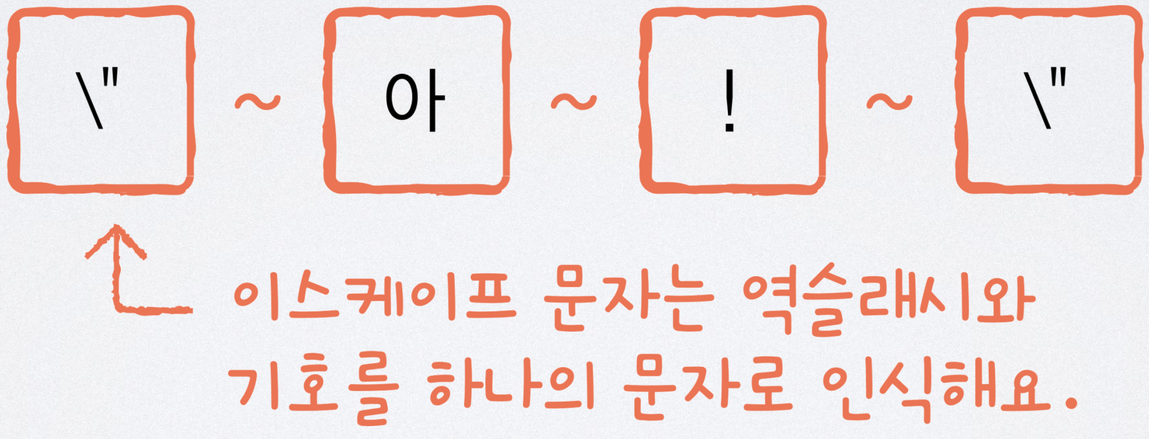
문자 데이터의 길이란 문자 데이터에 포함된 **문자의 개수**

`"Hello, World"`의 길이 → **12**
`"777910227236O7"`의 길이 → **14**
`"파이썬, 안녕?"`의 길이 → **8**
`"\"아!\""`의 길이 → **4**

문자 데이터의 길이를 계산할 때 **주의**할 점

- **공백**(space)도 한 개의 문자로 인식
- **특수문자**(?, ! 등)도 한 개의 문자로 인식
- **이스케이프 문자**도 한 개의 문자로 인식

이스케이프 문자는 역슬래시와 기호를 묶어서 한 개의 문자로 인식



파이썬 명령어 len은 문자 데이터의 길이를 구하는 명령어

```
len(문자  데이터)
```

파이썬 명령어 len은 문자 데이터의 길이를 구하는 명령어

```
len("Hello, World") → 12
len("777910227236O7") → 14
len("파이썬, 안녕?") → 8
len("\"아!\"") → 4
```

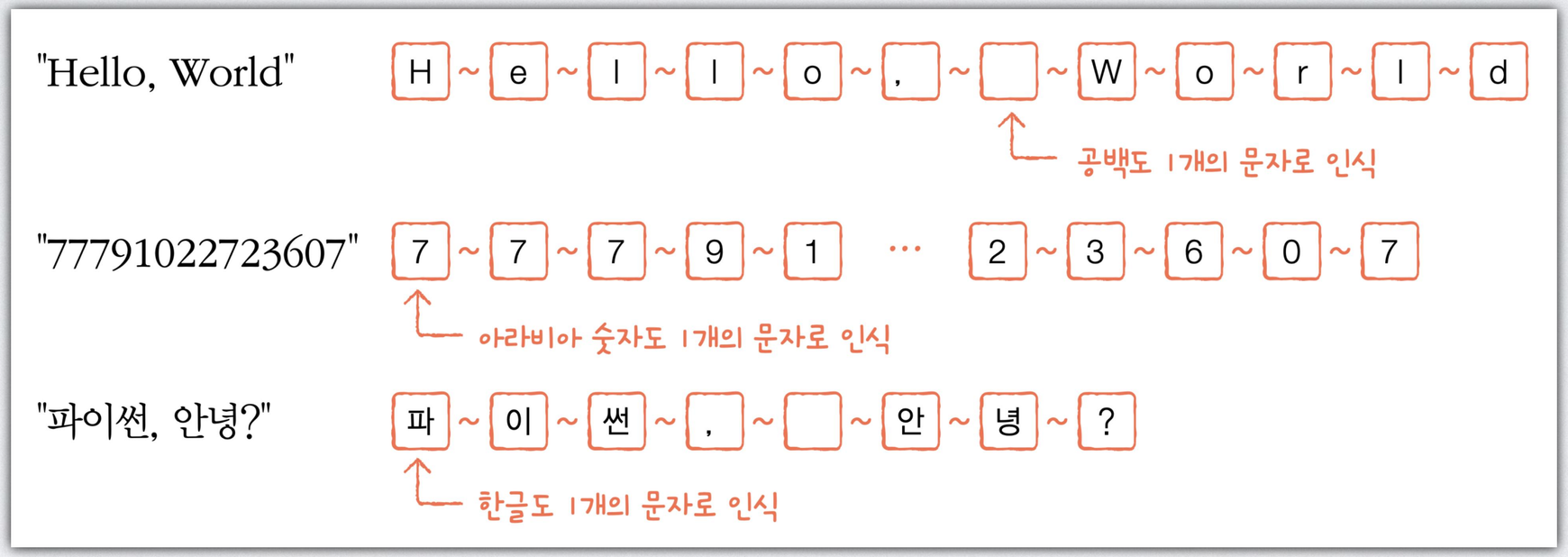
직접 해보는 손코딩

소스 코드 len01.py

```
01  print(len("AAA"))
02  print(len("Hello, World"))
03  print(len("77791022723607"))
04  print(len("파이썬, 안녕?"))
05  print(len("\"아!\""))
```

문자 데이터 **슬라이싱(자르기)**

문자 데이터는 **한 글자씩 순서대로** 메모리에 저장되고 관리됨



문자 데이터 **슬라이싱**(slicing, 자르기)란

자를 위치를 정해서 문자 데이터를 자르는 방법

1. **문자 데이터**를 준비하고,
2. **자를 위치**(시작과 끝)를 정하고,
3. **자를 도구**를 사용해서 문자 데이터를 자른다.

문자 데이터 **슬라이싱**(slicing, 자르기)란

자를 위치를 정해서 문자 데이터를 자르는 방법

문자 데이터[시작 위치:끝 위치]

문자 데이터에 저장된 각 문자에 **인덱스**(index)가 자동으로 부여됨

문자 데이터를 자를 **시작 위치** 및 **끝 위치**는 **인덱스**로 범위를 지정

인덱스는 0부터 시작

문자 데이터	K	E	9	0	1
인덱스	0	1	2	3	4

```
"KE901"[0:2] → "KE"

"KE901"[2:5] → "901"
```

프로그래밍 언어가 문자 데이터를 자를 때 **끝 위치 포함하지 않음!**

시작 위치와 **끝 위치**에 대한 **예외 사항**

1. **시작 위치 생략**하면, **처음부터** 자름
2. **끝 위치 생략**하면, **끝까지** 자름
3. 모두 **생략**하면, **전체 데이터**를 자름

시작 위치와 끝 위치에 대한 예외 사항

```
"KE901"[:2] → "KE"

"KE901"[2:] → "901"

"KE901"[:] → "KE901"
```

직접 해보는 손코딩 

소스 코드 sli01.py

```
01  print("KE901"[0:2])
02  print("KE901"[2:5])
03  print("KE901"[2:100])
04  print("KE901"[2:])
05  print("KE901"[:2])
06  print("KE901"[:])
```

문자 데이터 **인덱싱**

문자 데이터에 저장된 각 문자는 **인덱스**(index)로 관리됨

인덱싱(indexing)이란 인덱스에 위치한 **문자 한 개**를 뽑는 것

문자 데이터[문자 인덱스]

문자 데이터	K	E	9	0	1
인덱스	0	1	2	3	4

```
"KE901"[0] → "K"
"KE901"[1] → "E"
"KE901"[2] → "9"
"KE901"[3] → "0"
"KE901"[4] → "1"
```

직접 해보는 손코딩

소스 코드 ind01.py

```
01  print("KE901"[0])
02  print("KE901"[1])
03  print("KE901"[2])
04  print("KE901"[3])
05  print("KE901"[4])
```