

단원 04

문자열과 리스트

인공지능소프트웨어학과

강환수 교수



Section 1. 문자열

-



문자열 표현 방법

• 문자열 생성

- 작은따옴표(') 또는 큰따옴표(")로 묶어 문자열을 생성
 - `stmt1 = 'Explicit is better than implicit.'`
 - `stmt2 = "Simple is better than complex."`
- 따옴표 자체를 문자열 내에 포함
 - 서로 다른 종류의 따옴표를 사용하거나, 이스케이프 시퀀스(\)를 사용
 - `stmt1 = " 'Explicit' is better than implicit."`
 - `stmt2 = ' "Simple" is better than complex.'`
- 여러 줄 문자열
 - 삼중 따옴표(''') 또는 (""") 로 묶어 표현
 - `stmt3 = '''Flat is better than nested.
... Sparse is better than dense.'''`

• 문자열 연결

- + 연산자를 사용하여 문자열을 연결
 - 숫자형과 문자열을 연결할 때에는 `str()` 함수를 사용하여 숫자형을 문자열로 변환

문자열 관련 내장 함수

- **str(obj)**
 - 객체를 문자열로 변환
 - 주로 숫자형을 문자열로 변환할 때 사용
- **len(str)**
 - 문자열의 길이를 반환
- **chr(i)**
 - 정수 i에 해당하는 유니코드 문자를 반환
- **ord(c)**
 - 문자 c에 해당하는 유니코드 코드 값을 반환
- **min(iterable) / max(iterable)**
 - 문자열 내에서 최소/최대 코드 값을 갖는 문자를 반환

```
>>> min('python'), max('python')
('h', 'y')
>>> min('1234'), max('1234')
('1', '4')
```

이스케이프 시퀀스

escape sequence

• 역슬래시 \ 사용

- \\ (역슬래시), \' (작은따옴표), \" (큰따옴표) 등의 특수 문자를 표현하기 위해 사용

이스케이프 시퀀스 문자	설명	이스케이프 시퀀스 문자	설명
\\	역슬래시	\uxxxx	16비트 16진수 코드
\'	작은따옴표	\Uxxxxxxxx	32비트 16진수 코드
\"	큰따옴표	\ooo	8진수의 3자리 코드 문자
\a	벨소리(알람)	\xhh	16진수의 2자리 코드 문자
\b	백스페이스(이전 문자 지우기)	\N{name}	유니코드의 이름
\n	새 줄		
\r	보통 같은 줄에서 맨 앞으로 이동		
\f	폼피드(form feed) (예전에 프린터에서 다음 페이지 첫 줄로 이동)		
\t	수평 탭		
\v	수직 탭		

문자열 불변성

• 문자열

- 불변(immutable) 객체
- 문자열 메소드는 원본 문자열을 변경하지 않고 새로운 문자열을 반환

• 함수 vs 메소드

- 함수
 - 독립적으로 호출
- 메소드
 - 특정 객체에 속하여 객체를 통해 호출

TIP

도와주세요~ 용어가 어려워요!

함수와 메서드

함수(function)는 특정 작업을 수행하는 독립된 '코드 모임'이다. 함수 len()을 호출할 때 독립적으로 len('function') 이라고 호출한다. 반면, 메서드(method)는 클래스에 포함되어 있는 함수를 말한다. 메서드 호출을 살펴보면 'object'.count('o') 처럼 str 객체 'object'에 점(.)을 붙인 후 메서드 count('o') 이름으로 호출한다. 즉, 함수는 독립적으로 직접 호출하지만, 메서드는 객체를 통해 호출한다.

함수

len('python')

↓

독립적

특정한 업무를 수행
하는 프로그램 단위

메서드

'python'.count('th')

↓

객체 또는 클래스에 소속

▲ 그림 1 함수와 메서드

다양한 문자열 메소드 1/4

- **str 객체에서 사용할 수 있는 다양한 메소드가 제공**
 - capitalize, count, find, index, join, replace, split, strip, upper, lower, center
 - ljust, rjust, startswith, endswith
 - isalnum, isalpha, isdigit, islower, isnumeric, isprintable, isspace, istitle, isupper
- **help(str.capitalize) # capitalize 메소드 사용법 확인**
- **capitalize()**
 - 문자열의 첫 글자를 대문자로, 나머지를 소문자로 변환
- **split(sep=None, maxsplit=-1)**
 - 문자열을 공백 또는 지정된 구분자로 분리하여 리스트로 반환
- **join(iterable)**
 - 주어진 iterable 객체의 항목을 문자열로 연결

다양한 문자열 메소드 2/4

- **replace(old, new, count=-1)**
 - 문자열에서 지정된 old 문자열을 new 문자열로 변환
 - count를 지정하면 최대 count 개수만큼 변환
- **zfill(width)**
 - 지정된 너비만큼 0을 채워 문자열을 반환
- **count(sub)**
 - 문자열 내에서 지정된 sub 문자열의 출현 횟수를 반환

다양한 문자열 메소드 3/4

- **find(sub) / index(sub)**
 - 문자열 내에서 지정된 sub 문자열의 시작 위치를 반환
 - find()는 없을 경우 -1을 반환하고 index()는 ValueError를 발생
- **rfind(sub) / rindex(sub)**
 - 문자열 내에서 지정된 sub 문자열을 뒤에서부터 검색하여 시작 위치를 반환
- **upper() / lower()**
 - 문자열을 모두 대문자/소문자로 변환
- **swapcase()**
 - 문자열의 대문자는 소문자로, 소문자는 대문자로 변환
- **casefold()**
 - 문자열을 모두 소문자로 변환해 반환

다양한 문자열 메소드 4/4

- **title()**
 - 문자열 내 단어의 첫 글자를 대문자로, 나머지를 소문자로 변환
- **center(width, fillchar=' ')**
 - 문자열을 지정된 너비에 가운데 정렬하고, 나머지 공간을 fillchar로 채움
- **ljust(width, fillchar=' ') / rjust(width, fillchar=' ')**
 - 문자열을 지정된 너비에 왼쪽/오른쪽 정렬하고, 나머지 공간을 fillchar로 채움
- **strip(chars=None) / lstrip(chars=None) /rstrip(chars=None)**
 - 문자열의 양쪽/왼쪽/오른쪽에서 지정된 문자(기본적으로 공백)를 제거

Section 2. 리스트와 튜플

-



리스트(List) 정의

리스트는 여러 종류의 항목들을 순서대로 담을 수 있는 가변적인 시퀀스

- `[item_0, item_1, ..., item_n]` 형식으로 표현

- '항목의 모음(collections of items)'이고, 항목은 '순서가 있는 시퀀스(sequence)'
 - 목록이라고도 부름
- 항목은 원소(elements), 요소(items)라고도 부름
 - 항목은 모든 유형의 개체(object)가 가능

- 특징

- 가변성(Mutable): 리스트는 항목을 추가, 삭제, 변경 가능
- 다양한 자료형 저장: 리스트는 정수, 실수, 문자열, 다른 리스트 등 다양한 자료형의 항목을 저장 가능
- 중복 허용: 리스트 내에 동일한 값의 항목을 여러 개 가질 수 있음

- 생성:

- [] 리터럴: 직접 항목을 지정하여 생성

```
>>> data_list = [1, 2.4, 'py', [1, 2, 3], range(5)]

>>> coffee = ['에스프레소', '아메리카노', '카페라떼', '카페모카']

>>> coffeeprice = [['에스프레소', 2500], ['아메리카노', 2800], ['카페라떼', 3200]]

>>> program_languages = [('basic', 1964), ('java', 1995), ('python', 1991), ('c', 1972)]
```

내장 함수 list()

list() 함수를 사용하여 다른 이터러블(iterable) 객체를 리스트로 변환

- **list('Python')**
 - 문자열을 개별 문자 항목으로 분리한 리스트를 생성
 - ['P', 'y', 't', 'h', 'o', 'n']
- **list(range(10))**
 - 0부터 9까지의 정수를 항목으로 갖는 리스트를 생성
 - [0, 1, 2, ..., 8, 9]
- **빈 리스트**
 - list() 또는 []를 사용하여 빈 리스트를 생성

내장 함수 list(iterable)

`list([iterable])`

인자 iterable은 생략할 수 있다.

- 인자 iterable은 반복 가능한 객체이면 가능
- 인자 iterable이 없이 list()이면 빈 목록이 반환
- 인자 iterable로는 str, list, tuple, set, frozenset, dict, range 등이 가능

함수 insert() append()

- 항목 위치 추가 연산자 insert(index, item)

- 지정된 위치(index)에 항목을 삽입
- 메서드 insert(index, item)는 첨자 위치 index를 지정해 index 바로 앞에 항목 obj를 삽입

- 항목 마지막 추가 연산자 append(item)

- 리스트의 맨 마지막에 항목을 추가
- 메서드 append()는 무조건 맨 마지막에 인자 항목을 추가

```
>>> languages = ["Python", "Java", "C++", "Go"]
>>> languages.insert(2, 'JavaScript')
>>> languages
['Python', 'Java', 'JavaScript', 'C++', 'Go']

>>> languages.append('Kotlin')
>>> languages
['Python', 'Java', 'JavaScript', 'C++', 'Go', 'Kotlin']
```

함수 `extend()` `remove()` `pop()`

- `extend(iterable)`

- 다른 이터러블 객체의 모든 항목을 리스트의 맨 마지막에 추가
- `iterable`의 객체로 목록을 확장
- 인자 `iterable`은 반드시 반복 가능한 객체여야 함
 - `devices.extend(['laptop', 'tablet'])`과 같이 여러 항목을 추가
 - `devices.extend(range(3))`과 같이 `range` 객체를 이용하여 여러 정수를 추가
 - `devices.extend(100)`과 같이 정수는 이터러블 객체가 아니므로 `TypeError`가 발생

- 항목 삭제 연산자 `remove(item)`

- 리스트에서 지정된 값의 항목을 삭제
- 해당 값이 없으면 `ValueError`가 발생
 - `remove(obj)`는 목록에서 `obj`를 제거

- `pop(index)`

- 지정된 위치(`index`)의 항목을 삭제하고 그 항목 값을 반환
- 인덱스를 지정하지 않으면 마지막 항목을 삭제

- `del list[index]`

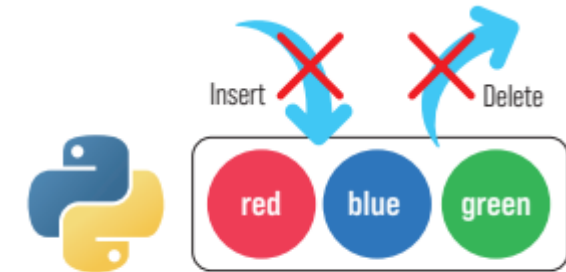
- `del` 키워드를 사용하여 리스트의 특정 위치의 항목을 삭제
- `del`은 메모리에서 객체를 삭제하며 삭제 후 객체를 참조하면 에러가 발생

튜플(Tuple) 정의

튜플은 여러 종류의 항목들을 순서대로 담을 수 있는 불변적인 시퀀스

- (item_0, item_1, ..., item_n) 형식으로 표현

- 튜플도 목록이나 문자열과 같은 시퀀스
- 튜플은 리스트와 다르게 항목 추가, 삭제가 불가능
- 튜플은 모두 괄호()로 구분된 항목들의 목록으로 표현
 - 각 항목은 정수, 실수, 문자열, 리스트, 튜플 등의 제한이 없음



튜플은 수정할 수 없어요!

튜플 tuple 형태	
(item_0, item_1, item_2, ..., item_n-1, item_n)	괄호 내부의 항목 나열은 '튜플'이다.
• item_i은 모든 유형의 객체가 가능하며 항목이 하나라도 콤마 필요	
item_0, item_1, item_2, ..., item_n-1, item_n	
• 괄호는 생략 가능하며 항목이 하나라도 콤마 필요	

튜플 생성

튜플은 여러 종류의 항목들을 순서대로 담을 수 있는 불변적인 시퀀스

- 특징
 - 불변성 (Immutable)
 - 튜플은 생성 후 항목을 추가, 삭제, 변경 불가능
 - 다양한 자료형 저장
 - 리스트와 마찬가지로 다양한 자료형의 항목을 저장 가능
 - 읽기 전용 데이터에 적합
 - 변경이 불필요한 데이터 집합을 저장할 때 유용
- 생성
 - 괄호 안에 항목들을 쉼표로 구분하여 생성
 - `season = ('봄', '여름', '가을', '겨울')`
 - 괄호 생략
 - 튜플의 항목이 하나인 경우, 괄호를 생략하면 해당 항목의 자료형으로 인식되므로 주의 필요
 - 튜플로 인식시키려면 반드시 쉼표를 포함

내장 함수 tuple()

- tuple() 함수
 - tuple() 함수를 사용하여 다른 이터러블 객체를 튜플로 변환
- tuple('python')
 - 문자열을 개별 문자 항목으로 분리한 튜플을 생성
- tuple(range(5))
 - 0부터 4까지의 정수를 항목으로 갖는 튜플을 생성
- 빈 튜플
 - tuple() 또는 () 를 사용하여 빈 튜플을 생성

패킹과 언패킹

- 패킹 (Packing)

- 여러 개의 값을 하나의 튜플로 묶는 것을 의미
 - `day = ('월', '화', '수')`

- 언패킹 (Unpacking)

- 튜플의 항목들을 여러 개의 변수에 각각 할당하는 것을 의미
- 이후에 '패킹된' 튜플은 다음처럼 튜플 항목 수와 같은 여러 변수에 나누어 대입
 - 이를 '언패킹(unpacked, 압축 해제)'이라고 함
 - `a, b, c = day`
- 언패킹 시 변수의 개수는 튜플의 항목 개수와 일치해야 함
 - 그렇지 않으면 `ValueError`가 발생
 - `a, b = day`

- 수정 불가능

- 튜플은 불변적이므로 항목을 추가나 삭제 불가능
 - 튜플의 항목을 변경(추가나 삭제)하려고 하면 `TypeError`가 발생
 - 하지만 튜플 내의 리스트 항목 자체의 수정은 가능

Section 3. 첨자와 슬라이싱

-

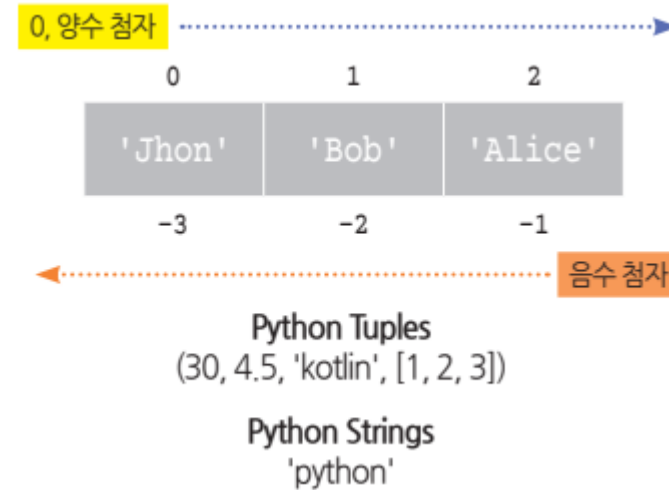
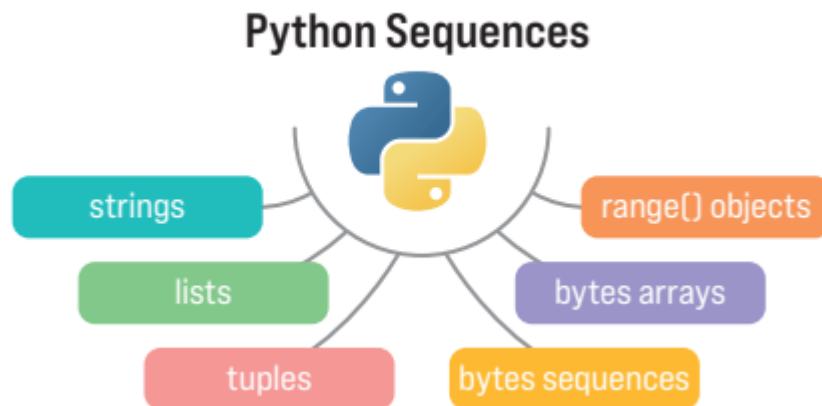


시퀀스 자료형 개요

자료 유형 str, list, tuple, range, bytearray, bytes 등이 시퀀스

정의

- 시퀀스는 데이터가 순서대로 나열된 자료 구조
 - 각 항목은 위치(index)로 참조 가능



시퀀스 자료형 개요

- 수정 가능 (mutable)
 - list, bytearray
- 수정 불가 (immutable)
 - str, tuple, bytes

	순서가 있는 항목 시퀀스	순서가 없는 항목 모음
수정 가능 (mutable)	list bytearray	set dict
수정 불가능 (immutable)	str tuple bytes	frozenset

- 순서가 있는 항목 시퀀스
 - 시퀀스는 위치에 따라 정렬된 항목 모음
 - 자료형 str, list, tuple, range, bytearray, bytes 등이 시퀀스
 - 시퀀스에서 항목은 순서가 있으며 첨자로 그 위치를 지정 가능
- 순서가 없는 항목 시퀀스
 - set, frozenset, dict 등은 순서가 없고 인덱스로 접근할 수 없는 자료형

bytes

- **bytes:**
 - 8비트 바이트의 연속 표현으로, 수정 불가능
- **bytes(n)**
 - 크기가 n이고 0으로 채워진 바이트 객체 생성
- **bytes(list)**
 - 리스트 항목으로 채워진 바이트 객체 생성
 - b'문자열'
 - b 접두사는 Python에서 문자열 리터럴이 바이트 데이터임을 표현
 - 문자열로 채워진 바이트 객체 생성

```
>>> bytes(5) # 정해진 길이만큼 0으로 채워진 바이트 객체를 생성
b'\x00\x00\x00\x00\x00'
>>> bytes([0, 0, 0, 0, 0]) # 리스트 항목으로 채워진 바이트 객체를 생성
b'\x00\x00\x00\x00\x00'
>>> bytes(b'python') # 지정한 문자열의 문자로 채워진 바이트 객체를 생성
b'python'

>>> b'hello'
b'hello'
>>> b'hello' == bytes([104, 101, 108, 108, 111])
True
```

bytes()와 bytearray()

- bytes()

- 수정할 수 없음
- ord() 함수를 사용하여 문자 코드로 변환 가능

- bytearray()

- 바이트 배열로, 수정 가능
 - 문자열을 bytes()를 사용하여 바이트 형태로 변환 가능
- 바이트 데이터를 decode() 메서드를 사용하여 문자열로 디코딩 가능

```
>>> b = bytes(b'python')
```

```
>>> b[1]
```

```
121
```

```
>>> ord('y')
```

```
121
```

```
>>> b = bytes(b'python')
```

```
>>> b[1]
```

```
121
```

```
>>> ord('y')
```

```
121
```

```
>>> b[0] = ord('P')
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'bytes' object does not support item assignment
```

```
>>> ba = bytearray(b'python')
```

```
>>> ba
```

```
bytearray(b'python')
```

```
>>> ba[0]
```

```
112
```

```
>>> ba[0] = ord('P')
```

```
>>> ba
```

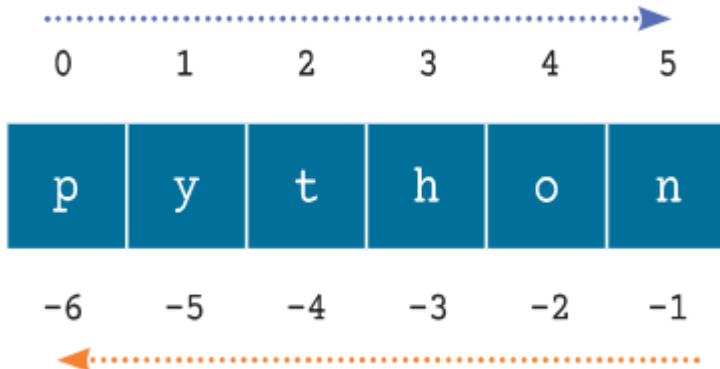
```
bytearray(b'Python')
```

```
>>> []
```


첨자(Index) 참조

- 시퀀스 객체에서 `sequence_obj[index]` 형식으로 항목을 참조 가능
 - 인덱스는 0부터 시작하며, 마지막 요소는 -1로 참조 가능
 - 인덱스 범위 초과 시 `IndexError` 발생
- 주의
 - 문자열 및 튜플은 수정 불가능한 객체이므로, 인덱스를 통한 항목 수정은 불가능

0, 양수 첨자: 0 ~ [len(시퀀스)-1]



음수 첨자: [-len(시퀀스) ~ -1]

```
>>> p = 'python'
>>> p[0]
'p'
>>> p[5]
'n'
>>> p[-6]
'p'
>>> p[-1]
'n'
```

시퀀스에서 첨자로 항목 수정

- 두 번째 항목 수정

```
>>> # 수정 가능 객체
>>> season = ['봄', '여름', '가을', '겨울']
>>> season[1] = 'summer'
>>> season
['봄', 'summer', '가을', '겨울']
```

- 수정 불가능한 객체인 문자열이나 튜플은 다음처럼 수정이 불가능

```
>>> # 수정 불가능한 객체
>>> p = 'python'
>>> p[0] = 'p'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> goods = ('볼펜', '만년필')
>>> goods[1] = '연필'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

count()와 index() 메서드

- **count(obj):**
 - 시퀀스 내에서 특정 객체 obj의 출현 횟수를 반환
- **index(obj):**
 - 시퀀스 내에서 특정 객체 obj의 위치를 반환

```
>>> cities = ["New York", "Los Angeles", "Chicago", "New York"]
>>> cities.count('New York')
2
>>> cities.index('Los Angeles')
1
>>> seasons = ("봄", "여름", "가을", "겨울", "겨울")
>>> seasons.count('겨울')
2
>>> seasons.index('가을')
2
```

슬라이싱(Slicing)

시퀀스의 일부분을 추출하는 기능

- **sequence_obj[start:stop:step] 형식으로 사용**
 - start: 슬라이스의 시작 인덱스 (기본값 0)
 - stop: 슬라이스의 끝 인덱스 (stop은 포함하지 않음, 기본값은 시퀀스의 끝)
 - step: 슬라이스 간격 (기본값 1)

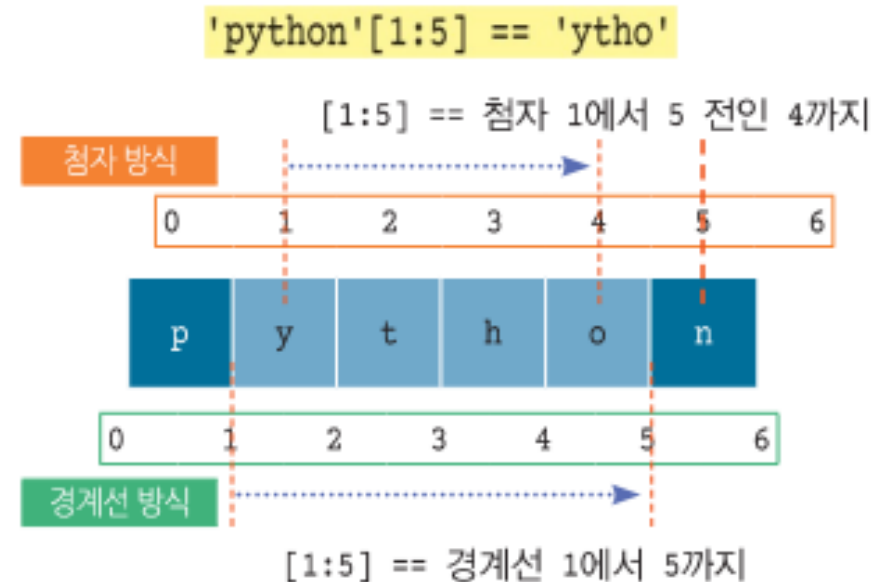


- **슬라이싱 결과**
 - 슬라이싱은 새로운 시퀀스 객체를 반환
- **슬라이스를 이용한 요소 수정**
 - 슬라이스를 사용하여 리스트의 일부 요소를 변경

첨자를 사용한 시퀀스 슬라이싱

- `sequence_obj[start:stop:step]`

```
>>> 'python'[1:5]
'ytho'
>>> 'python'[2:4]
'th'
>>> 'python'[0:3]
'pyt'
>>> 'python'[0:6]
'python'
>>> 'python'[0:len('python')]
'python'
```

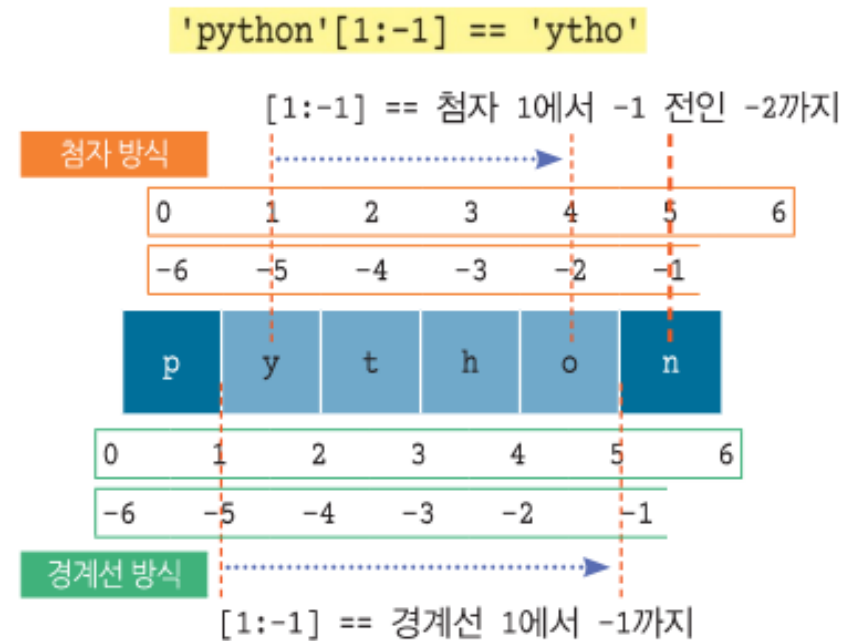


▲ 그림 4 0과 양수를 이용한 슬라이싱

음수를 이용한 슬라이싱

- 음수와 양수, 함께 사용도 가능

```
>>> 'python'[1:-1]
'ytho'
>>> 'python'[0:-2]
'pyth'
>>> 'python'[2:-2]
'th'
>>> 'python'[-5:5]
'ytho'
>>> 'python'[-6:4]
'pyth'
>>> 'python'[-4:6]
'thon'
```

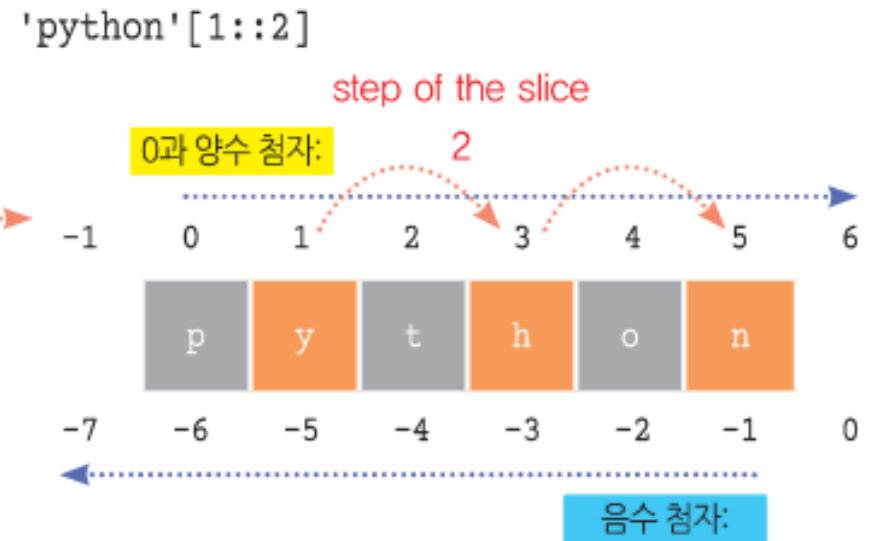


▲ 그림 6 음수 0, 양수를 사용한 슬라이싱

인자 step으로 항목 사이의 간격을 조정

- 시퀀스 슬라이싱에서 항목 사이의 간격을 step으로 지정해 조절
 - 마지막 인자 step이 2이면 하나씩 건너뛰는 부분 슬라이싱이 반환

```
>>> 'python'[::-2]
'pto'
>>> 'python'[1::2]
'yhn'
>>> 'python'[1:5:2]
'yh'
>>> 'python'[1:5:3]
'yo'
```

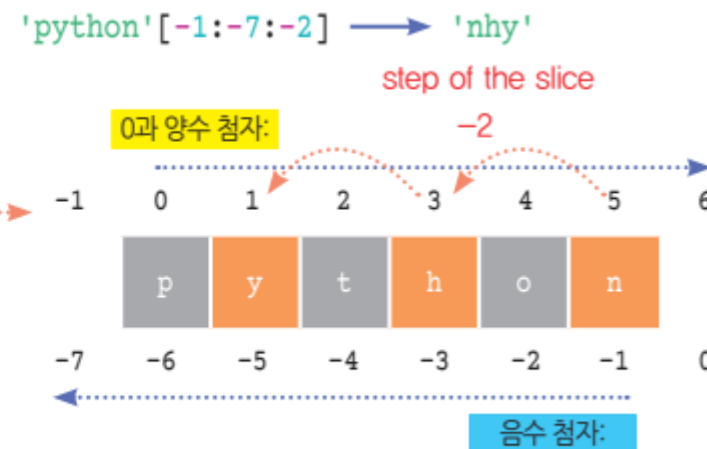


▲ 그림 7 간격 step을 사용한 슬라이싱

step이 음수이면 시퀀스의 반대 방향

- 마지막 인자인 step이 양수
 - 첫 인자 start는 두 번째 stop보다 왼쪽에 위치하고 있는 첨자
- 간격 step이 음수
 - start는 stop보다 오른쪽에 위치하고 있는 첨자

```
>>> 'python'[5:0:-1]
'nohty'
>>> 'python'[-1:-7:-2]
'nhy'
>>> 'python'[-1:-7:-1]
'nohtyp'
```



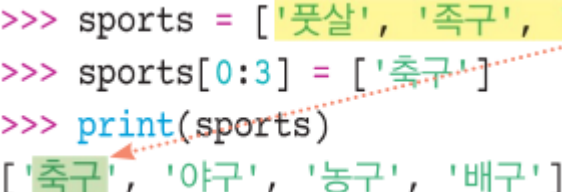
▲ 그림 8 시퀀스의 반대 방향으로 시퀀스 구성

```
>>> 'python'[::-1] # 역순의 문자열을 반환
'nohtyp'
```

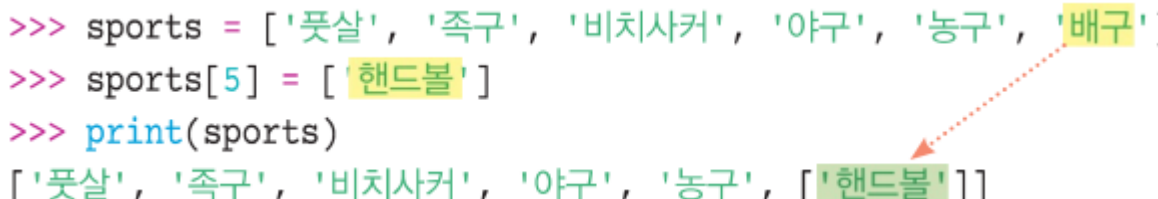

슬라이스를 사용하여 리스트의 일부 요소를 수정

- 리스트의 항목 또는 부분 리스트를 수정

```
>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> sports[0:3] = ['축구']
>>> print(sports)
['축구', '야구', '농구', '배구']
```



```
>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> sports[5] = ['핸드볼']
>>> print(sports)
['풋살', '족구', '비치사커', '야구', '농구', ['핸드볼']]
```



```
>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> sports[5] = '핸드볼'
>>> print(sports)
['풋살', '족구', '비치사커', '야구', '농구', '핸드볼']
```

리스트 슬라이스에 슬라이스 대입

	첨자	0	1	2	3	4	5
--	----	---	---	---	---	---	---

```

>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> sports[1:3] = sports[4:6]
>>> print(sports)
['풋살', '농구', '배구', '야구', '농구', '배구']
  
```

'족구', '비치사커'가 '농구', '배구'로 대체된다.

```

>>> sports = ['풋살', '족구', '비치사커', '야구', '농구', '배구']
>>> others = ['축구', '미식축구', '골프']
>>> sports[1:4] = others[:2]
>>> print(sports)
['풋살', '축구', '미식축구', '농구', '배구']
  
```

시퀀스 연산자 + *

- + (연결 연산자)

- 시퀀스를 연결하여 새로운 시퀀스를 생성
 - 리스트와 리스트를 연결하여 새로운 리스트를 생성
 - 튜플과 튜플을 연결하여 새로운 튜플을 생성
 - 문자열과 문자열을 연결하여 새로운 문자열을 생성

- * (반복 연산자)

- 시퀀스를 반복하여 새로운 시퀀스를 생성

```
>>> 'java' + "python"
'javapython'

>>> season1 = ('봄', '여름')
>>> season2 = ('가을', '겨울')
>>> season1 + season2
('봄', '여름', '가을', '겨울')

>>> 'py' * 3
'pypypy'

>>> 4 * "ja"
'jajajaja'
```

연결 대입 연산자 +=

• 리스트에 += 연산자를 사용

- 원본 리스트가 수정
 - id 값 유지

```
>>> colors = ["blue", "indigo"]
>>> id(colors)
1936281589952
>>> colors += ["violet"]
>>> colors
['blue', 'indigo', 'violet']
>>> id(colors)
1936281589952
```

수정 가능한 자료형 리스트는 수정 전후의 주소 값이 같다.

• 문자열 또는 튜플에 += 연산자를 사용

- 새로운 객체가 생성
 - id 값 변경

```
>>> s = 'py'
>>> id(s)
140709189531232
>>> s += 'thon'
>>> s
'python'
>>> id(s)
1936281947632
```

수정 불가능한 자료형 문자열은 수정 전후의 주소 값이 다르다.

소속 연산자 in

멤버십 연산자 in

- in (소속 연산자)

- 특정 값이 시퀀스에 포함되어 있는지 여부를 확인
 - 값이 포함되어 있으면 True, 아니면 False 반환
- 문자열, 리스트, 튜플, 딕셔너리, 집합, range 객체에 사용 가능

멤버십 연산자 in
value [not] in collection
<ul style="list-style-type: none"> • in: value가 모음인 collection의 내부에 있으면 True, 없으면 False 반환 • not in: value가 모음인 collection 내부에 없으면 True, 있으면 False 반환

- not in

- in 반대

```
>>> 'beautiful' not in 'Flat is better than nested.'
True
>>> '봄봄' not in ('봄', '여름', '가을', '겨울')
True
>>> 'june' not in ['may', 'september', 'august']
True
```

리스트의 얕은 복사

• 얕은 복사 (Shallow Copy)

- 리스트를 대입 연산자로 대입하면, 두 변수는 같은 리스트 객체를 참조 (id 값 동일)
- 리스트의 요소 변경 시 대입된 변수에도 동일하게 반영

```
>>> org_lst = ['사과', '귤', '배']
>>> asmt_lst = org_lst
>>> id(org_lst) == id(asmt_lst)
True
```

두 변수는 같은 객체가 된다.

```
>>> org_lst.pop()
'배'
>>> asmt_lst, org_lst
(['사과', '귤'], ['사과', '귤'])

>>> # 튜플의 얕은 복사
>>> org_tpl = ('봄', '여름', '가을', '겨울')
>>> asmt_tpl = org_tpl
>>> id(org_tpl) == id(asmt_tpl)
True
```

리스트 깊은 복사

• 깊은 복사 (Deep Copy)

- 슬라이싱 [:], list(), copy() 메서드 등을 사용하여 새로운 리스트 객체를 생성 (id 값 변경)
- 원본 리스트 변경 시 복사된 리스트에 영향을 주지 않음

```
>>> org_lst = ['사과', '귤', '배']
>>> cp_lst = org_lst[:]
>>> id(org_lst) == id(cp_lst)
False
>>> org_lst.pop(2)
'배'
>>> org_lst, cp_lst
(['사과', '귤'], ['사과', '귤', '배'])
```

두 변수는 다른 객체가 된다.

튜플 얹은 복사

- 튜플

- 수정할 수 없는 튜플에는 깊은 복사가 없으며
 - `asmt_tpl = org_tpl[:]`으로도 구 변수가 동일한 튜플을 가리킴

- 연산자 `is`

- 피연산자인 변수 2개가 동일한 메모리를 공유하는지를 검사
 - 튜플이 같으면 `True`, 다르면 `False`를 반환

```
>>> org_tpl = ('시', '분', '초')
>>> asmt_tpl = org_tpl[:]
>>> asmt_tpl = tuple(org_tpl)
>>> org_tpl is asmt_tpl
True
```