05

Data Types

Python built-in objects

Object type	Example literals/creation
Numbers	1234, 3.1415, 0b111, 1_234, 3+4j, Decimal, Fraction
Strings	'code', "app's", b'a\x01c', 'h\u00c4ck', 'hÄck'
Lists	[1, [2, 'three'], 4.5], list(range(10))
Tuples	(1, 'app', 4, 'U'), tuple('hack'), namedtuple
Dictionaries	{'job': 'dev', 'years': 40}, dict(hours= 10)
Sets	set('abc'), {'a', 'b', 'c'}
Files	open('docs.txt'), open(r'C:\data.bin', 'wb')
Other core objects	Booleans, types, None
Program-unit objects	Functions, modules, classes
Implementation objects	Compiled code, stack tracebacks

Mark Lutz. (2025). Learning Python, 6th Edition. n.p.: O'Reilly Media, Inc.

List

- 여러 값을 순서대로 저장하는 자료형
- 대괄호 [] 사용, 각 요소는 쉼표로 구분

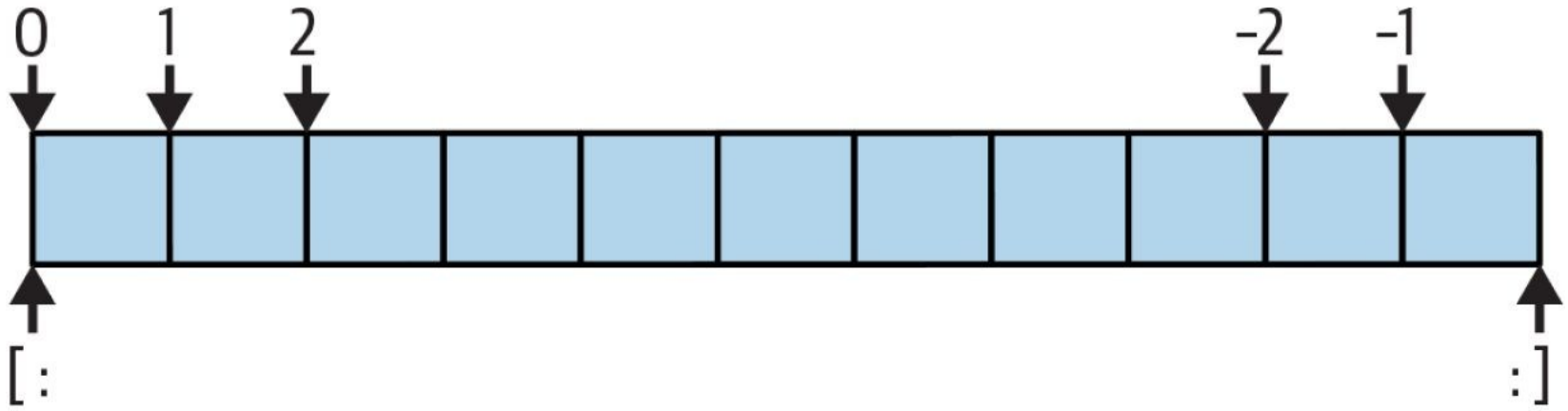
```
fruits = ['apple', 'banana', 'cherry']
```

리스트의 특징

- 순서 있음 (indexing 가능)
- 중복된 값 허용
- 다양한 자료형 혼합 가능
- 변경 가능 (mutable)

Indexing

```
>>> L = [123, 'text', 1.23]    # A list of three different-type objects
>>> L[0]                       # Indexing by position (offset)
123
>>> L[:-1]                     # Slicing a list returns a new list
[123, 'text']
```



Slicing

```
>>> L = ['hack', 'Hack', 'HACK!']
>>> L[2]                          # Offsets start at zero
'HACK!'
>>> L[-2]                         # Negative: count from the right
'Hack'
>>> L[1:]                         # Slicing fetches sections
['Hack', 'HACK!']

>>> L[1] = ‘code’                 # Index assignment
>>> L
['hack', 'code', 'HACK!']
>>> L[0:2] = [‘Python’, ‘write’]    # Slice assignment: "delete+insert"
>>> L
['Python', 'write', 'HACK!']
```

Slicing

- 슬라이싱 범위에 따라 교체/삽입/삭제 가능

```
>>> L = [1, 2, 3]
>>> L[1:2] = [4, 5]              # Replacement/insertion
>>> L
[1, 4, 5, 3]
>>> L[1:1] = [6, 7]              # Insertion (replace nothing)
>>> L
[1, 6, 7, 4, 5, 3]
>>> L[1:2] = []                  # Deletion (insert nothing)
>>> L
[1, 7, 4, 5, 3]
```

Operators

- +: 리스트 연결, *: 리스트 반복
- in: 포함 여부 확인

```
>>> len([1, 2, 3])                  # Length
3
>>> [1, 2, 3] + [4, 5, 6]           # Concatenation
[1, 2, 3, 4, 5, 6]
>>> ['Py!'] * 4                     # Repetition
['Py!', 'Py!', 'Py!', 'Py!']

>>> str([1, 2]) + '34'               # Same as '[1, 2]' + '34'
'[1, 2]34'
>>> [1, 2] + list('34')              # Same as [1, 2] + ['3', '4']
[1, 2, '3', '4']
```

Nested List

- 중첩 리스트: 리스트 안에 리스트를 요소로 포함한 구조
- 2차원 배열의 경우, 행과 열의 형태로 데이터 표현 가능

```
>>> matrix = [[1, 2, 3],
...           [4, 5, 6],
...           [7, 8, 9]]
>>> matrix[0]
[1, 2, 3]                  # <class 'list'>
>>> matrix[1][2]
6                          # <class 'int'>
```

Adding elements to list

- append(): 맨 끝에 추가
- insert(index, value): 원하는 위치에 삽입
- extend(): 여러 요소 확장

```
>>> L = ['write']
>>> L.append('Python')                  # Append: add one item at the end
>>> L
['write', 'Python']
>>> L.insert(2, 'code')                 # Insert: add value at index
>>> L
['write', 'Python', 'code']
>>> L.extend(['clean', '!'])             # Extend: add many items at the end
>>> L
['write', 'Python', 'code', 'clean', '!']
```

Eliminating elements from list

- remove(value): 값 제거 (첫 번째 항목)
- pop(index): 인덱스로 제거, 기본은 마지막
- del a[index]: 리스트 a에서 인덱스의 요소값 삭제
- clear(): 전체 요소 제거

```
>>> L = [1, 2, 3, 4, 5, 6]
>>> L.remove(2)                  # L = [1, 3, 4, 5, 6]
>>> L.pop()
6                                # L = [1, 3, 4, 5]
>>> L.pop(0)
1                                # L = [3, 4, 5]
>>> del L[1]                     # L = [3, 5]
>>> L.clear()
>>> L
[]
```

Sorting, etc.

- sort(): 오름차순 정렬
- reverse(): 반대 순서로 뒤집기 (정렬X)
- index(value): 위치 찾기
- count(value): 등장 횟수 세기

```
>>> L = [4, 2, 7, 2, 9, 2]
>>> L.count(2)
3
>>> L.index(7)
2
>>> L.sort()
>>> L
[2, 2, 2, 4, 7, 9]
>>> L.reverse()
>>> L
[9, 7, 4, 2, 2, 2]
```

Sorting, etc.

- sort()는 원본 리스트를 정렬(in-place)
- sorted()는 정렬된 새 리스트를 반환하며, 원본 리스트는 변하지 않음

```
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort()
>>> L
['ABD', 'aBe', 'abc']
>>> L = ['abc', 'ABD', 'aBe']
>>> L.sort(reverse=True)
>>> L
['abc', 'aBe', 'ABD']
```

```
>>> L = ['abc', 'ABD', 'aBe']
>>> sorted(L)
['ABD', 'aBe', 'abc']
>>> L                    # L is not modified
['abc', 'ABD', 'aBe']
```

Copy / Reference

- a = b는 같은 리스트를 함께 참조
- a = b.copy()는 같은 내용의 리스트를 생성(얕은 복사)

```
>>> a = [1, 2, 3]
>>> b = a
>>> b[0] = 100
>>> a
[100, 2, 3]
>>> b
[100, 2, 3]
```

```
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> b[0] = 100
>>> a
[1, 2, 3]
>>> b
[100, 2, 3]
```

Tuple

- 리스트와 유사
- 소괄호 () 사용, 각 요소는 쉼표로 구분

```
fruits = ('apple',)                    # 요소의 개수가 1개라면 요수 뒤에 콤마(,) 필요
fruits = ('apple', 'banana', 'cherry')
```

튜플의 특징

- 순서 있음 (indexing 가능)
- 중복된 값 허용
- 다양한 자료형 혼합 가능
- **변경 불가능 (immutable)**

변경 불가능 (immutable)

- 튜플의 요소값은 한 번 정하면 지우거나 변경 불가능

```
>>> T = (1, 2, 3, 'a', 'b')
>>> del T[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
>>> T[0] = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Indexing, Slicing

- 문자열, 리스트와 마찬가지로 인덱싱, 슬라이싱 가능

```
>>> T = (1, 2, 3, 'a', 'b')
>>> T[2]
3
>>> T[1:]
(2, 3, 'a', 'b')
>>> T[::-1]
('b', 'a', 3, 2, 1)
```

Operators

- 문자열, 리스트와 마찬가지로 튜플끼리 ‘+’, ‘*’ 연산 가능
  산 가능

```
>>> T1 = (1, 2, 3, 'a', 'b')
>>> T2 = (4, 5)
>>> T1 + T2
(1, 2, 3, 'a', 'b', 4, 5)
>>> T2 * 3
(4, 5, 4, 5, 4, 5)
```

Other methods

- index(value): value의 인덱스를 반환, 튜플 내에 없으면 오류 발생
- count(value): value가 튜플 내에 몇 번 포함되어 있는지 반환

```
>>> T = (1, 2, 3, 4, 4)
>>> len(T)
5
>>> T.index(4)
3
>>> T.count(4)
2
```

5주차 과제

- 05_Data_types.ipynb