03. 데이터 다루기 – 리스트, 튜플, 딕셔너리

❖ 리스트

- 모든 객체의 시퀀스
- 리스트 메소드
- [ ]

❖ 튜플

- 패킹과 언패킹
- 튜플 메소드
- ( )

❖ 딕셔너리

- 키와 값 으로 구성
- { }

리스트

- 리스트(List)는 이름에서 알 수 있듯이 데이터의 목록을 다루는 자료형
- 단일 데이터가 명함이라면, 리스트는 명함을 모아두는 명함집

- 슬롯(Slot) : 리스트의 데이터를 삽입할 자리
- 요소(Element) : 리스트의 각 슬롯에 꽂혀있는 개별 데이터

- C/C++나 자바 프로그래밍 언어에서의 배열(Array)과 비슷한 개념
- 차이점 : 서로 다른 데이터형도 요소가 될 수 있다는 점

- 다차원 리스트도 당연히 존재

리스트

- **리스트 생성 : 대괄호 [ ] 또는 list( ) 사용**
- **실습 1 (문자열 입력)**

```
>>> a = ['김개똥', '박짱구', '이멍충']
>>> a
['김개똥', '박짱구', '이멍충']
>>> a[0]
'김개똥'
>>> a[1]
'박짱구'
>>> a[2]
'이멍충'
```

리스트를 만들 때는 대괄호 [와] 사이에 데이터 또는 변수 목록을 입력.
각 데이터는 콤마(,)로 구분.

리스트는 문자열처럼 참조 연산이 가능.
리스트 이름 뒤에 대괄호를 붙이고 [와] 사이에 참조하고자 하는 첨자를 입력.

- **실습 2 (수 입력)**

```
>>> a = [1, 2, 3, 4]
>>> a
[1, 2, 3, 4]
```

리스트

❖ 실습 3 (슬라이싱)

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> a[0:5]
[1, 2, 3, 4, 5]
>>> a[5:]
[6, 7, 8, 9, 10]
>>> a[:3]
[1, 2, 3]
```

❖ 실습 4 (+ 연산자를 통한 리스트간의 결합)

```
>>> a = [1, 2, 3, 4]
>>> b = [5, 6, 7]
>>> a + b
[1, 2, 3, 4, 5, 6, 7]
```

리스트

❖ 실습 5 (리스트 내의 특정 위치에 있는 데이터를 변경)

```
>>> a = [1, 2, 3, 4, 5]
>>> a[2] = 30
>>> a
[1, 2, 30, 4, 5]
>>> a[3] = 40
>>> a
[1, 2, 30, 40, 5]
```

❖ 실습 6 (len() 함수로 리스트 길이 재기)

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

리스트 - 메소드

메소드	설명

append()	리스트의 끝에 새 요소를 추가합니다.
	(다른 타입의 요소를 포함)
	(항목 자체가 하나의 인자로 추가)
	
	>>> a = [1, 2, 3]
	>>> a.append([4, 5, 6])
	>>> a
	[1, 2, 3, [4, 5, 6]]

extend()	기존 리스트에 다른 리스트를 이어 붙입니다. + 연산자와 같은 기능을 한다고 할 수 있습니다.
	(다른 리스트를 병합)
	
	>>> a = [1, 2, 3]
	>>> a.extend([4, 5, 6])
	>>> a
	[1, 2, 3, 4, 5, 6]

insert()	첨자로 명시한 리스트 내의 위치에 새 요소를 삽입합니다. insert(첨자, 데이터)의 형식으로 사용합니다.
	
	>>> a = [2, 4, 5]
	>>> a.insert(0, 1) # 0 위치(첫 번째)에 데이터 1을 삽입합니다.
	>>> a
	[1, 2, 4, 5]
	>>> a.insert(2, 3) # 2 위치(세 번째)에 데이터 3을 삽입합니다.
	>>> a
	[1, 2, 3, 4, 5]

리스트 - 메소드

메소드	설명

remove()	매개 변수로 입력한 데이터를 리스트에서 찾아 발견한 첫 번째 요소를 제거합니다.
	
	>>> a = ['BMW', 'BENZ', 'VOLKSWAGEN', 'AUDI']
	>>> a.remove('BMW')
	>>> a
	['BENZ', 'VOLKSWAGEN', 'AUDI']

pop()	리스트의 마지막 요소를 뽑아내어 리스트에서 제거합니다.
	
	>>> a = [1, 2, 3, 4, 5]
	>>> a.pop()
	5
	>>> a
	[1, 2, 3, 4]
	>>> a.pop()
	4
	>>> a
	[1, 2, 3]
	
	한편, 마지막이 아닌 특정 요소를 제거하고 싶을 때에는 pop() 메소드에 제거하고자 하는 요소의 인덱스를 입력하면 됩니다.
	>>> a = [1, 2, 3, 4, 5]
	>>> a.pop(2) # 3번째 요소 제거
	3
	>>> a
	[1, 2, 4, 5]

리스트 - 메소드

메소드	설명

index()	리스트 내에서 매개변수로 입력한 데이터와 일치하는 첫번째 요소의 첨자를 알려줍니다. 찾고자 하는 데이터와 일치하는 요소가 없으면 오류를 일으킵니다. 오류에 대한 처리 방법은 10장에서 설명하겠습니다.
	
	>>> a = ['abc', 'def', 'ghi']
	>>> a.index('def')
	1
	>>> a.index('jkl')
	Traceback (most recent call last):
	  File "<pyshell#2>", line 1, in <module>
	  a.index('jkl')
	ValueError: 'jkl' is not in list

count()	매개변수로 입력한 데이터와 일치하는 요소가 몇 개 존재하는지 셉니다.
	
	>>> a = [1, 100, 2, 100, 3, 100]
	>>> a.count(100)
	3
	>>> a.count(200)
	0

리스트 - 메소드

메소드	설명

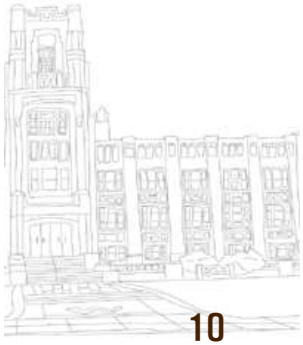
sort()	리스트 내의 요소를 정렬합니다. 매개변수로 reverse = True를 입력하면 내림차순, 아무 것도 입력하지 않으면 오름차순으로 정렬합니다. reverse = True와 같이 이름을 명시하여 사용하는 매개변수를 일컬어 키워드 매개변수라고 합니다. 키워드 매개변수는 7장에서 자세히 설명합니다.
	
	>>> a = [3, 4, 5, 1, 2]
	>>> a.sort()
	>>> a
	[1, 2, 3, 4, 5]
	>>> a.sort(reverse = True)
	>>> a
	[5, 4, 3, 2, 1]

reverse()	리스트 내 요소의 순서를 반대로 뒤집습니다.
	
	>>> a = [3, 4, 5, 1, 2]
	>>> a.reverse()
	>>> a
	[2, 1, 5, 4, 3]
	>>> b = ['안', '녕', '하', '세', '요']
	>>> b.reverse()
	>>> b
	['요', '세', '하', '녕', '안']

튜플

- **사전적 의미는 Tuple과 List가 비슷**
  - 리스트는 “목록”
  - 튜플은 “N개의 요소로 된 집합”

- **파이썬의 List와 Tuple의 차이**
  - List는 데이터 변경 가능(리스트 생성 후 추가/수정/삭제 가능)
  - Tuple은 데이터 변경 불가능(튜플 생성 후 추가/수정/삭제 불가능)
  - List는 이름 그대로 목록 형식의 데이터를 다루는 데 적합
  - Tuple은 위경도 좌표나 RGB 색상처럼 작은 규모의 자료구조를 구성하기에 적합



튜플

❖ 변경이 불가능한 자료형이 필요한 이유?

- 성능
  - 변경 가능한 자료형과는 달리 데이터를 할당할 공간의 내용이나 크기가 달라지지 않기 때문에 생성 과정이 간단
  - 데이터가 오염되지 않을 것이라는 보장이 있기 때문에 복사본을 만드는 대신 그냥 원본을 사용
- 신뢰 가능한 코드
  - 변경되지 않아야 할 데이터를 오염시키는 버그를 만들 가능성 제거
  - 코드를 설계할 때부터 변경이 가능한 데이터와 그렇지 않은 데이터를 정리해서 코드에 반영

❖ 문자열도 변경이 불가능한 자료형

튜플

❖ 실습 1 (튜플 생성)

```
>>> a = (1, 2, 3) # []가 아닌 ()를 사용
>>> a
(1, 2, 3)
>>> type(a)
<class 'tuple'>
```

❖ 실습 2 (튜플 생성2)

```
>>> a = 1, 2, 3, 4 # () 없이 콤마(,) 만 사용
>>> a
(1, 2, 3, 4)
>>> type(a)
<class 'tuple'>
```

튜플

❖ 실습 3 (요소가 하나인 튜플 생성)

```
>>> a = (1,) # 요소가 하나인 경우엔 요소 뒤에 , 추가
>>> a
(1,)
>>> type(a)
<class 'tuple'>
>>> b = 1, # 요소가 하나인 경우엔 요소 뒤에 , 추가
>>> b
(1,)
>>> type(b)
<class 'tuple'>
```

튜플

❖ 실습 4 (슬라이싱)

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> a[:3]
(1, 2, 3)
>>> a[4:6]
(5, 6)
```

❖ 실습 5 (+ 연산자를 이용한 튜플간 결합)

```
>>> a = (1, 2, 3)
>>> b = (4, 5, 6)
>>> c = a + b
>>> a
(1, 2, 3)
>>> b
(4, 5, 6)
>>> c
(1, 2, 3, 4, 5, 6)
```

튜플

❖ 실습 6 (변경 불가능 테스트)

```
>>> a = (1, 2, 3)
>>> a[0]
1
>>> a[0] = 7
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    a[0] = 7
TypeError: 'tuple' object does not support item assignment
```

❖ 실습 7 (len() 함수)

```
>>> a = (1, 2, 3)
>>> len(a)
3
```

튜플 - 패킹과 언패킹

❖ **실습 1 (튜플 패킹(Tuple Packing))**

```
>>> a = 1, 2, 3 # 패킹 : 여러 데이터를 튜플로 묶는 것
>>> a
(1, 2, 3)
```

❖ **실습 2 (튜플 언패킹(Tuple Unpacking))**

```
>>> one, two, three = a # 언패킹 : 튜플의 각 요소를
>>> one                 # 여러 개의 변수에 할당하는 것.
1
>>> two
2
>>> three
3
```

튜플 - 패킹과 언패킹

❖ **실습 3 (언패킹 실패)**

```
>>> a = 1, 2, 3 # 튜플 요소 수와
>>> one, two = a # 언패킹할 요소의 수가 불일치
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    one, two = a
ValueError: too many values to unpack (expected 2)
```

❖ **실습 4 (언패킹을 이용한 변수 다중 할당)**

```
>>> city, latitude, longitude = 'Seoul', 37.541, 126.986
>>> city
'Seoul'
>>> latitude
37.541
>>> longitude
126.986
```

'Seoul', 37.541, 126.986는 괄호 없이 만들어진 튜플

튜플 - 패킹과 언패킹

❖ 실습 5 (언패킹을 이용한 변수 다중 할당)

```
>>> p = 'abc'
>>> q = 'defg'
>>> p, q = q, p
>>> p
 'defg'
>>> q
 'abc'
```

치환문 오른쪽이 듀플로 패킹 된 후에,
치환문 왼쪽의 변수에 언팩킹 되어 할당.
(임시변수를 사용 하지 않고 교환 됨)

❖ 실습 6 (확장 언패킹)

```
>>> a = (1, 2, 3, 4, 5)
>>> b, *c, d = a
>>> print(a)
(1, 2, 3, 4, 5)
>>> print(b, c, d)
1  [2, 3, 4]  5        # 확장 언패킹(*)은 리스트
```

튜플 - 메소드

메소드	설명

index()	매개변수로 입력한 데이터와 일치하는 튜플 내 요소의 첨자를 알려줍니다. 찾고자 하는 데이터와 일치하는 요소가 없으면 에러를 일으킵니다. 에러에 대한 처리 방법은 6장에서 설명하겠습니다.
	
	>>> a = ('abc', 'def', 'ghi')
	>>> a.index('def')
	1
	>>> a.index('jkl')
	Traceback (most recent call last):
	  File "<pyshell#4>", line 1, in <module>
	  a.index('jkl')
	ValueError: tuple.index(x): x not in tuple

count()	매개변수로 입력한 데이터와 일치하는 요소가 몇 개 존재하는지 셉니다.
	
	>>> a = (1, 100, 2, 100, 3, 100)
	>>> a.count(100)
	3
	>>> a.count(200)
	0

딕셔너리

- **딕셔너리(Dictionary)는 사용법 측면으로 보면 리스트와 비슷**
  - 리스트처럼 첨자를 이용해서 요소에 접근
- **리스트는 요소에 접근할 때 0부터 시작하는 수 첨자만 사용할 수 있지만 딕셔너리는 문자열과 숫자를 비롯해서 변경이 불가능한 형식이면 어떤 자료형이든 키로 사용**
  - 딕셔너리의 첨자는 키(Key) - 반드시 유일
  - 이 키가 가리키는 슬롯에 저장되는 데이터를 일컬어 값(Value)

  ➔딕셔너리는 키-값의 쌍으로 구성

- **탐색 속도가 빠르고, 사용하기도 편리**
- **딕셔너리를 만들 때는 중괄호 {와 }을 이용**
- **특정 슬롯에 새로운 키-값을 입력하거나 딕셔너리 안에 있는 요소를 참조할 때는 리스트와 튜플에서처럼 대괄호 [와]를 이용**

딕셔너리

❖ 실습 1 (딕셔너리 생성)

```
>>> dic = {}
>>> dic['파이썬'] = 'www.python.org'
>>> dic['마이크로소프트'] = 'www.microsoft.com'
>>> dic['애플'] = 'www.apple.com'
>>> dic['파이썬']
'www.python.org'
>>> dic['마이크로소프트']
'www.microsoft.com'
>>> dic['애플']
'www.apple.com'
>>> type(dic)
<class 'dict'>
>>> dic
{'애플': 'www.apple.com', '파이썬': 'www.python.org', '마이크로
소프트': 'www.microsoft.com'}
```

딕셔너리

❖ 실습 2 (딕셔너리의 keys(), values() 메소드)

```
>>> dic.keys()
dict_keys(['애플', '파이썬', '마이크로소프트'])
>>> dic.values()
dict_values(['www.apple.com', 'www.python.org', 'www.microsoft.com'])
```

❖ 실습 3 (딕셔너리의 items() 메소드)

```
>>> dic.items()
dict_items([('파이썬', 'www.python.org'), ('애플', 'www.apple.com'), ('
마이크로소프트', 'www.microsoft.com')])
```

❖ 실습 4 (in 연산자)

```
>>> '애플' in dic.keys()
True
>>> '사과' in dic.keys()
False
>>> 'www.microsoft.com' in dic.values()
True
>>> 'www.seanlab.net' in dic.values()
False
```

'애플'이 dic 딕셔너리의 키 목록 안에 존재하는지를 확인합니다.

'www.seanlab.net'이 dic 딕셔너리의 값 목록 안에 존재하는지를 확인합니다.

딕셔너리

❖ 실습 5 (딕셔너리의 pop() 메소드)

```
>>> dic = {'애플': 'www.apple.com',
        '파이썬': 'www.python.org',
        '마이크로소프트': 'www.microsoft.com'}
>>> dic.pop('애플')
'www.apple.com'
>>> dic
{'파이썬': 'www.python.org', '마이크로소프트': 'www.microsoft.com'}
```

키가 '애플'인 요소을 삭제합니다.

❖ 실습 6(딕셔너리의 clear() 메소드)

```
>>> dic = {'애플': 'www.apple.com',
        '파이썬': 'www.python.org',
        '마이크로소프트': 'www.microsoft.com'}
>>> dic.clear()
>>> dic
{}
```

셋

❖ 값은 버리고 키만 남은 딕셔너리 (존재 여부 판단에 사용)

```
>>> empty_set = set()
>>> empty_set
Set()  # { }을 출력 하지 않음

>>> numbers = {1, 2, 3, 4}
>>> numbers
{2, 4, 3, 1}
```

❖ 리스트, 문자열, 듀플, 딕셔너리로 부터 중복된 값을 제거한 셋을 생성
딕셔너리에 set( )을 사용 하면 키에만 적용

```
>>> set( 'letters' )
{l,e,t,r,s}
```

중요 사항

❖ 복합 객체 자료형 (복합 시퀀스 객체의 트리)

- 내장된 자료형을 결합해서 더 크고 복잡한 자료구조 생성

```
>>> L = [ ‘abc’,  [ (1, 2), ([3], 4) ],   5 ]
>>> L[1]
[ (1, 2), ([3], 4) ]
>>> L[1][1]
([3], 4)
>>> L[1][1][0]
[3]
>>> L[1][1][0][0]
3
```

❖ 할당 : =

```
>>> a = [ 1, 2, 3 ]
>>> b = a                    # 얇은 복사
>>> a[0] = ‘surprise’
>>> a
[ ‘surprise, 2, 3 ]
>>> b
[ ‘surprise, 2, 3 ]
```

중요 사항

❖ 복사 : copy( )

```
>>> a = [ 1, 2, 3 ]
>>> b = a.copy( )                  # 깊은 복사    copy 모듈 사용시  b=copy.copy(a)
>>> c = list(a)                    # 깊은 복사
>>> d = a[ : ]                     # 깊은 복사
>>> a[0] = ‘surprise’              # b, c, d 는 변동 없음
```

❖ 재귀복사 : deepcopy( )- 복합 객체를 재귀적으로 생성해서 복사

```
>>> a = [ 1, 2, 3 ]
>>> b = [ 4, 5, 6 ]
>>> c = [ a, b, 100 ]
>>> d = c.deepcopy( )     # copy 모듈 사용시  d=copy.deepcopy(c)
```

❖ 저장 모델에 따른 자료형의 분류

- 리터럴(Literal) – int, float, complex, bool, byte, str
  - 동일한 값들은 서로 같은 주소
    - 리터럴
- 저장(Container) – list, tuple, dict, set
  - 동일한 값들 이라도 서로 다른 주소 (별도로 만들어 짐)
    - 리터럴들을 별도의 공간에 저장