변수와 자료형 (3)

PDF	비어 있음
비고	리스트, 디큐, 튜플
숫자	5
실습문제	비어 있음
실습문제답안	비어 있음

03. 리스트 (List)

03-01. 리스트 개요

03-01-01. 리스트란

💡 일련의 값이 모인 집합을 다루기 위한 자료형으로 python은 배열과 같은 표현식을 갖지만, 일반적인 프로그래밍 언어와 다르게 길이를 동적으로 조절할 수 있어 list라고 부른다.

03-01-02. 리스트 표현식

```python
리스트명 = ['값1', '값2', ... ]
```

- List의 각 요소의 자료형은 무엇이든 될 수 있으며, 서로 다른 자료형이어도 괜찮다.
- List에 저장되는 요소들은 0부터 시작하는 인덱스 체계로 구분해서 저장된다.

```python
fruits = ['orange', 'apple','pear','banana','kiwi', 'apple'] print(fruits) print(fruits[0])
```

03-01-03. 문자열의 split()

- split()을 사용하여 문자열을 구분자 기준으로 분리한 List로 만들 수 있다.

```python
rainbow_split = '빨-주-노-초-파-남-보' print(rainbow_split.split('-'))
```

03-01-04. list() 메서드

- list() 메서드로도 List를 만들 수 있다.

```python
list1 = [] list2 = list() print(list1 == list2) # True print(type(list1)) # <class 'list'> print(type(list2)) # <class 'list'>
```

```python
list3 = list('hello') # only 1 argument print(list3) # ['h', 'e', 'l', 'l', 'o']
```

03-02. 리스트 연산

03-02-01. 리스트 합치기: +

- 리스트 간 + 연산자를 이용하여 하나로 합칠 수 있다.

```
safari_list = ["Bear", "Koara", "Gorilla", "Squirrel"] another_safari_list = ["Monke
y", "Tiger", "Wolf"] print(safari_list + another_safari_list) # ['Bear', 'Koara', 'Go
rilla', 'Squirrel', 'Monkey', 'Tiger', 'Wolf']
```

03-02-02. 리스트 반복: *

- 리스트에 * 연산을 하면 요소를 반복할 수 있다.

```
safari_list = ["Bear", "Koara", "Gorilla", "Squirrel"] print(safari_list * 4) # ['Bea
r', 'Koara', 'Gorilla', 'Squirrel', 'Bear', 'Koara', 'Gorilla', 'Squirrel', 'Bear',
'Koara', 'Gorilla', 'Squirrel', 'Bear', 'Koara', 'Gorilla', 'Squirrel']
```

03-03. 리스트 메서드

- 리스트 메서드 확인을 위해 사용하는 fruits 배열을 다음과 같이 먼저 선언해둔다.

```
fruits = ['orange', 'apple','pear','banana','kiwi', 'apple']
```

03-03-01. count()

- count(값)은 해당 리스트에 인자로 준 값이 몇 개 존재하는지 확인하여 그 수를 반환한다.

```
print("apple:", fruits.count('apple')) # apple: 2
```

- 만약 해당 값이 존재하지 않는다면 0이 반환된다.

```
print("test:", fruits.count("test")) # test: 0
```

03-03-02. index()

- index(값)은 리스트에서 인자로 준 값이 몇 번째 인덱스에 존재하는지 확인하여 그 인덱스를 반환한다.
- 같은 값이 리스트 내에 여러 개 존재하면 가장 처음에 등장하는 값의 인덱스를 반환한다.

```
print("index:", fruits.index("apple")) # index: 1
```

- 만약 해당 값이 존재하지 않는다면 에러가 발생한다.

```
# print("index:", fruits.index("apple1")) # error 발생
```

- index(값, 인덱스)와 같이 매개변수를 2개 전달할 수 있다.
  - 첫 번째 매개변수 '값': 인덱스를 찾을 대상 값
  - 두 번째 매개변수 '인덱스': 첫 번째 매개변수로 준 값을 두 번째 매개변수로 준 인덱스 이후 위치에서 탐색

```
print("index:", fruits.index("apple", 3)) # index: 5
```

03-03-03. reverse()

- reverse()는 list의 값을 역으로 정렬한다.

```
fruits.reverse() print(fruits) # ['apple', 'kiwi', 'banana', 'pear', 'apple', 'orang
e']
```

03-03-04. append()

- append(값)은 list에 값을 덧붙여 추가한다.

```
fruits.append("pineapple") print(fruits) # ['apple', 'kiwi', 'banana', 'pear', 'appl
e', 'orange', 'pineapple']
```

03-03-05. extend()

- extend(값)은 list에 배열로 요소를 덧붙여 추가한다.

```
fruits.extend(['rainbow friut', 'sweet lemon', 'happy apple']) print(fruits) # ['appl
e', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'pineapple', 'rainbow friut', 'sweet
lemon', 'happy apple']
```

03-03-06. insert()

- insert(index, 값)은 list의 원하는 index 위치에 요소를 추가한다.

```
fruits.insert(0, 'monkey banana') print(fruits) # ['monkey banana', 'apple', 'kiwi',
'banana', 'pear', 'apple', 'orange', 'pineapple', 'rainbow friut', 'sweet lemon', 'ha
ppy apple']
```

03-03-07. sort()

- 요소를 정렬하는 메서드로 원본 list에 영향을 준다.
- 기본적으로 숫자 오름차순, 알파벳의 첫 글자를 기준으로 오름차순 정렬한다.

```
fruits.sort() print(fruits) # ['apple', 'apple', 'banana', 'happy apple', 'kiwi', 'mo
nkey banana', 'orange', 'pear', 'pineapple', 'rainbow friut', 'sweet lemon']
```

- 매개변수로 다양한 옵션을 주어 다른 기준으로 정렬할 수 있다.

```
# 내림차순 정렬 fruits.sort(reverse=True) print(fruits) # ['sweet lemon', 'rainbow fri
ut', 'pineapple', 'pear', 'orange', 'monkey banana', 'kiwi', 'happy apple', 'banana',
'apple', 'apple']
```

```
# 문자열 길이가 짧은 순으로 정렬 fruits.sort(key=len) print(fruits) # ['pear', 'kiwi',
'apple', 'apple', 'orange', 'banana', 'pineapple', 'sweet lemon', 'happy apple', 'rai
nbow friut', 'monkey banana']
```

```
# 복합 기준 정렬 # 문자열 길이와 알파벳 순으로 복합 기준 정렬 fruits.sort(key=lambda x:
(len(x),x)) print(fruits) # ['kiwi', 'pear', 'apple', 'apple', 'banana', 'orange', 'p
ineapple', 'happy apple', 'sweet lemon', 'monkey banana', 'rainbow friut']
```

03-03-08. remove()

- remove(값)은 값에 해당하는 요소를 삭제한다.
- 이때 해당 요소가 여러 개라면 index가 앞쪽인 요소 한 개만 삭제한다.

```
temp = fruits.remove('kiwi') print(temp) # None (해당 메소드 결과로 반환되는 건 없음) p
rint(fruits) # ['pear', 'apple', 'apple', 'banana', 'orange', 'pineapple', 'happy app
le', 'sweet lemon', 'monkey banana', 'rainbow friut']
```

03-03-09. pop()

- 마지막 요소를 추출하면서 제거한다.

```
temp = fruits.pop() print(temp) # rainbow friut print(fruits) # ['pear', 'apple', 'ap
ple', 'banana', 'orange', 'pineapple', 'happy apple', 'sweet lemon', 'monkey banana']
```

03-04. 리스트를 이용한 자료구조

03-04-01. Stack

- stack 자료구조는 FILO(first-in last-out)의 특징을 가진다.
- 자료를 쌓아올리므로 가장 처음에 쌓은 자료는 가장 마지막에 꺼낼 수 있다.
  - 예) 접시 쌓기, 편의점 과자 진열대 등

- 프로그래밍에서 Stack을 사용하는 가장 대표적인 예시는 함수 호출 스택이 있다.

  → A 함수가 B 함수를 호출하는 경우, B 함수가 종료되어야 A 함수가 다음 작업을 이어갈 수 있는 식이다.
- stack을 구현하는 코드에는 append(), pop()이 있다.

```
plates = ['plate1', 'plate2'] plates.append('plate3') plates.append('plate4') print(p
lates) # ['plate1', 'plate2', 'plate3', 'plate4']
```

```
print(plates.pop()) # plate4 print(plates) # ['plate1', 'plate2', 'plate3'] print(pla
tes.pop()) # plate3 print(plates.pop()) # plate2 print(plates) # ['plate1']
```

03-05. del 키워드

03-05-01. del

- del 키워드를 통해 원본 배열의 일부 요소 또는 전체 목록을 제거할 수 있다.

```
abclist = ['A','B','C','D','E','F'] print(abclist) # ['A', 'B', 'C', 'D', 'E', 'F'] d
el abclist[0] # 0번째 인덱스의 요소 제거 print(abclist) # ['B', 'C', 'D', 'E', 'F'] de
l abclist[1:3] # 1번째 인덱스에서 3번째까지 제거 print(abclist) # ['B', 'E', 'F'] del
abclist[:] # 모든 요소 제거 print(abclist) # []
```

```
del abclist # abclist를 제거 # print(abclist) # list 자체를 제거했으므로 Error 발생
```

03-06. 디큐 (deque)

03-06-01. 디큐(dobule-ended queue, 양방향 큐) 란

- 파이썬의 collections 모듈에서 제공하는 자료구조로, 양쪽 끝에서 효율적인 삽입과 삭제가 가능하게 설계되었다.
- deque는 stack과 queue의 기능을 모두 지원하며, 특정한 상황에서는 list 보다 더욱 효율적으로 사용할 수 있다.

```
from collections import deque dq = deque() dq.append("a") dq.append("b") dq.append
("c") print(dq) # deque(['a', 'b', 'c'])
```

03-06-02. 디큐의 메서드

- append(), pop(): list에서의 append(), pop()과 동일하다.
- appendleft() : 왼쪽 끝에 요소를 추가한다.

```
dq.appendleft('x') print(dq) # deque(['x', 'a', 'b', 'c'])
```

- popleft() : 왼쪽 끝 요소 값 반환하며 제거한다.

```
value = dq.pop() print(value) # c left_value = dq.popleft() print(left_value) # x pri
nt(dq) # deque(['a', 'b'])
```

[추가] 리스트 표현 심화 (내포)

1. 리스트 내포 기본 표현식

```
# 1~10까지 제곱값을 담은 리스트 만들기 lst = [] for n in range(1,11): lst.append(n **
2) print(lst) # 리스트 내포 (List Comprehension) lst = [n*n for n in range(1,11)]
print(lst)
```

2. 조건이 있는 리스트 내포

```
# 1~10까지 홀수의 제곱값만 구하기 # n % 2 == 1인 경우만 True lst = [n * n for n in
range(1,11) if n % 2] print(lst)
```

3. 중첩 리스트 내포

```
[expression for expr1 in sequence1 for expr2 in sequence2 if condition]
```

- expression은 반드시 하나의 값이어야 한다. (여러 개일 경우 tuple|list로 작성한다.)

```
# 구구단 요소(tuple(dan,su,dan * su))로 리스트 만들기 _99table = [(dan, n, dan * n)
for dan in range(2, 10) for n in range(1, 10)] print(_99table)
```

```
_99table = [(dan, n, dan * n) for dan in range(2, 10) for n in range(1, 10) if dan ==
n] print(_99table)
```

```
# 2개의 sequence자료형에 대해서 카테시안곱으로 튜플 만들기 seq1 = 'abc' seq2 = (1,2,3)
print([(x,y) for x in seq1 for y in seq2]) # print([[x,y] for x in seq1 for y in
seq2])
```

```
# 실습문제 : Life is too short, you need python에서 aeiou제거하기 # 방법1 s = 'Life is
too short, you need python' print([x for x in s if(x not in ['a','e','i','o','u'])])
# 방법2 vowels = 'aeiou' print(''.join([a for a in s if a not in vowels]))
```

```
s = '아버지가 방에 화분을 들고, 신발을 신고 들어가신다.' stopwords = ['은', '는', '이',
'가', '을', '를', '에', '에게', '에서', ' ', '.', ','] print(s.split(' ')) print([w
for w in s if w not in stopwords]) for w in s.split(' '): for stp_wrd in stopwords:
if w.endswith(stp_wrd): w = w.replace(stp_wrd, '') print(w)
```

```
s = '아버지가 방에 화분을 들고, 신발을 신고 들어가신다.' stopwords = ['은', '는', '이',
'가', '을', '를', '에', '에게', '에서', ' ', '.', ','] [ w.replace(stp_wrd, '') for w
in s.split(' ') for stp_wrd in stopwords if w.endswith(stp_wrd) ]
```

---

04. 튜플 (Tuple)

04-01. 튜플 개요

04-01-01. 튜플이란

💡 여러 개의 값을 하나의 데이터 구조로 묶어서 관리할 수 있는 불변(immutable) 시퀀스이다. 리스트와 유사하지만, 한번 생성되면 각 요소 값을 생성하거나 수정, 삭제할 수 없다. 주로 데이터의 집합을 안전하게 유지하거나, 함수에서 여러 값을 반환할 때 사용된다.

04-01-02. 튜플 표현식

- 튜플의 요소는 소괄호()를 중첩으로 감싸 표현할 수 있다.

```
튜플명 = ('값1', '값2', ... )
```

```
tuples = (1, 2, 'hello'), ('test', 1,2,3,4) print(tuples) # ((1, 2, 'hello'), ('tes
t', 1, 2, 3, 4)) print(tuples[0]) # (1, 2, 'hello') print(tuples[0][0]) # 1
```

```
no_wrap_tuples = 1, 2, 3 # 소괄호로 감싸지 않아도 튜플로 생성
```

04-02. 튜플의 연산

04-02-01. 불변 객체

- 튜플은 불변 객체이므로 요소 값을 한번 할당하면 추가, 수정, 삭제가 불가하다.

```
tuples = (1, 2, 'hello'), ('test', 1,2,3,4) # tuples[0] = (1,2,3,4) # Error 발생
```

- 이때 값을 재할당 하면 값이 다시 입력될 수도 있는데, 이는 기존의 변수에 값이 재할당 되는 것이 아닌 새로운 변수가 생성되는 것이다.

```
testList = (1,2,3,4) print(id(testList)) # 1952413712848 testList = (3,4,5) print(id
(testList)) # 1952407991360
```

  - id() : 인자로 전달된 개체의 주소값을 반환한다.
    - 서로 다른 주소값을 가진다는 것은 새로운 변수가 생성된 것임을 의미한다.

04-02-02. 튜플 간 연산

- 튜플 요소의 변경이 아닌 튜플 간 연산은 가능하다.

```
# 튜플 합치기 another_safari_tuple = ("Monkey", "Tiger", "Wolf") print(safari_tuple +
another_safari_tuple) # ('Bear', 'Koara', 'Gorilla', 'Squirrel', 'Monkey', 'Tiger',
'Wolf') # 튜플 반복 print(safari_tuple * 3) # ('Bear', 'Koara', 'Gorilla', 'Squirre
l', 'Bear', 'Koara', 'Gorilla', 'Squirrel', 'Bear', 'Koara', 'Gorilla', 'Squirrel') #
튜플의 길이 print(len(safari_tuple + another_safari_tuple)) # 7
```

[참고] 튜플의 활용

- 튜플을 활용한 복수 개의 자료 할당

```
x,y,z = 1,2,3 print(x, y, z)
```

- 튜플을 이용한 값 치환

```
x,y = y,x print(x, y)
```

  - 3개 이상도 가능

```
x,y,z = y,z,x print(x, y, z)
```

05. 시퀀스 자료형 (Sequence type)

05-01. 시퀀스 자료형 개요

05-01-01. 시퀀스 자료형이란

💡 시퀀스 자료형은 여러 값을 순서대로 저장하는 데이터 구조이다. 대표적인 시퀀스 자료형에는 **리스트, 튜플, 문자열**이 있다. 시퀀스 자료형은 각 요소에 인덱스(index)를 가지고 있어, 이를 사용해 접근할 수 있다.

- 참고
  - string : 문자열 값을 가지는 자료형
  - list : 여러 값을 담을 수 있는 가변형 자료형
  - tuple : 여러 값을 담을 수 있는 불변형 자료형

```
# str name = "ohgiraffers" print(type(name)) # <class 'str'> # list list = [0, 1, 2, 3] print(type(list)) # <class 'list'> # tuple tuple = (1, 2, 3) print(type(tuple)) # <class 'tuple'>
```

  - type() : 인자로 넘어온 값의 자료형을 반환하는 메서드

05-01-02. 인덱싱

- 시퀀스 자료형의 인덱스를 가지고 특정 요소를 추출할 수 있다.

```
# 문자열 address = '대한민국 서울시 서초구' print(address[5]) # 서 print(address[9]) # 서 # 리스트 location = ['서울특별시', '부산광역시', '인천광역시', '광주광역시', '울산광역시'] print(location[2]) # 인천광역시 # 튜플 nation = ('대한민국', '미국', '중국', '일본', '프랑스') print(nation[3]) # 일본
```

- 인덱스를 음수로 지정하면 문자열 끝에서부터 요소를 찾는다.

```
# 문자열 address = '대한민국 서울시 서초구' print(address[-3]) # 서 print(address[-7]) # 서 # 리스트 location = ['서울특별시', '부산광역시', '인천광역시', '광주광역시', '울산광역시'] print(location[-2]) # 광주광역시 # 튜플 nation = ('대한민국', '미국', '중국', '일본', '프랑스') print(nation[-3]) # 중국
```

05-01-03. 슬라이싱

- 인덱스를 사용하여 기존 시퀀스 자료에서 원하는 만큼의 문자열을 추출할 수 있다.

```
# 문자열 address = '대한민국 서울시 서초구' print(address[5:8]) # 서울시 (5번 인덱스부터 8번 인덱스 전까지) # 리스트 location = ['서울특별시', '부산광역시', '인천광역시', '광주광역시', '울산광역시'] print(location[2:4]) # ['인천광역시', '광주광역시'] (2번 인덱스부터 4번 인덱스 전까지) # 튜플 nation = ('대한민국', '미국', '중국', '일본', '프랑스') print(nation[0:3]) # ('대한민국', '미국', '중국') (0번 인덱스부터 3번 인덱스 전까지)
```

- 자료형 뒤에 대괄호를 붙이고 *(시작 인덱스) : [(끝 인덱스)] : [(오프셋)]* 으로 문자열을 추출할 수 있다.
  - 시작 인덱스: 문자열에서 추출할 내용이 시작하는 인덱스 (시작 인덱스를 비우면 처음부터)
  - 끝 인덱스: 문자열에서 추출할 내용의 끝 글자 다음 인덱스 (끝 인덱스를 비우면 끝까지)
  - 오프셋: 시작 인덱스부터 오프셋 숫자만큼 더한 인덱스 위치마다에서 추출

```
# 문자열 address = '대한민국 서울시 서초구' print(address[9:]) # 서초구 (9번 인덱스부터 끝까지) print(address[-3:]) # 서초구 (뒤에서 3번째 인덱스부터 끝까지) print(address[1:12:4]) # 한서서 (1번 인덱스부터 12번 인덱스 전까지 추출하는데 4개씩 건너뛰면서 추출) print(address[::-1]) # 구초서 시울서 국민한대 (슬라이싱을 이용해 [::-1]을 주면 문자열 뒤집기) # 리스트 location = ['서울특별시', '부산광역시', '인천광역시', '광주광역시', '울산광역시'] print(location[2:]) # 2번 인덱스부터 끝까지 (['인천광역시', '광주광역시', '울산광역시']) print(location[-2:]) # 뒤에서 2번째 인덱스부터 끝까지 (['광주광역시', '울산광역시']) print(location[1:3:2]) # 1번 인덱스부터 3번 인덱스 전까지 추출하는데 2개씩 건너뛰면서 추출 (['부산광역시']) print(location[::-1]) # 슬라이싱을 이용해 [::-1]을 주면 뒤집기 (['울산광역시', '광주광역시', '인천광역시', '부산광역시', '서울특별시']) # 튜플 nation = ('대한민국', '미국', '중국', '일본', '프랑스') print(nation[1:]) # 1번 인덱스부터 끝까지 (('미국', '중국', '일본', '프랑스')) print(nation[-3:]) # 뒤에서 3번째 인덱스부터 끝까지 (('중국', '일본', '프랑스')) print(nation[1:5:2]) # 1번 인덱스부터 5번 인덱스 전까지 추출하는데 2개씩 건너뛰면서 추출 (('미국', '일본')) print(nation[::-1]) # 슬라이싱을 이용해 [::-1]을 주면 뒤집기 (('프랑스', '일본', '중국', '미국', '대한민국'))
```

05-02. 시퀀스 자료형의 연산 및 내장 함수

05-02-01. in

- '값 in 시퀀스 자료형'의 형태로 사용하며, 우항의 시퀀스 자료형에 좌항의 값이 포함되어 있으면 True를 반환한다.
  - 문자열에 대한 in 연산은 우항의 문자열에 좌항의 문자열이 순서에 맞게 포함된 경우 True를 반환한다.
  - 리스트와 튜플에 대한 in 연산은 좌항의 요소를 우항의 자료에 대한 '하나의 요소'로 보기 때문에 인덱싱 또는 슬라이싱한 형태로 데이터를 주면 안된다. 요소 자체로 in 을 판단하기 때문이다.

```
# 문자열 address = '대한민국 서울시 서초구' print('대한민국' in address) # True print('국민한대' in address) # False print('서울시서초구' in address) # False # 리스트 location = ['서울특별시', '부산광역시', '인천광역시', '광주광역시', '울산광역시'] print('광주광역시' in location) # True print(['광주광역시', '부산광역시'] in location) # False print(['서울특별시', '부산광역시'] in location) # False # 튜플 nation = ('대한민국', '미국', '중국', '일본', '프랑스') print('미국' in nation) # True print(('대한민국', '미국') in nation) # False
```