파이썬 프로그래밍

4. 리스트와 딕셔너리

00

Contents

❖ 수업 목표

- 리스트 프로그램을 작성할 수 있다.
- 튜플 프로그램을 작성할 수 있다.
- 딕셔너리 프로그램을 작성할 수 있다.

❖ 세부 목표

00

1. 리스트

❖ 리스트 (List)

- 여러 값을 함께 모을 수 있는 자료형

코드 3-1 여러 개의 변수에 사탕을 저장하는 코드

```
>>> candy0 = '딸기맛'
>>> print(candy0)
딸기맛
>>> candy1 = '레몬맛'
>>> print(candy1)
레몬맛
>>> candy2 = '수박맛'
>>> print(candy2)
수박맛
>>> candy3 = '박하맛'
>>> print(candy3)
박하맛
>>> candy4 = '우유맛'
>>> print(candy4)
우유맛
```

❖ 리스트 기본 구조

- [값1, 값2, 값3, …]
- 대괄호와 콤마 사용하여 만든다

코드 3-2 여러 개의 사탕을 하나의 변수에 저장하는 코드

```
>>> candies = ['딸기맛', '레몬맛', '수박맛', '박하맛', '우유맛']
>>> print(candies)
['딸기맛', '레몬맛', '수박맛', '박하맛', '우유맛']
```

00

1. 리스트

❖ 리스트 특징

- 순서가 의미 있음
- 서로 다른 자료형을 모을 수 있음
- 여러 값을 모아 하나의 변수명으로 처리 가능

syntax : 리스트

변수명 = [item1, item2, item3....]	lista = ['list', 1, 0.7]

syntax : 정확한 코딩

변수명 = [item1, item2, item3....]	• 리스트를 정의할 때 대입 연산자 오른쪽에 각 요소를 '[' , ']'(꺽쇄 괄호)로 묶어주면 된다. • item(요소)은 모든 데이터형이 올 수 있다. • 각 요소는 쉼표(,)로 구분한다.

00

1. 리스트

❖ 리스트 요소(element)

- 리스트를 구성하는 각각의 값
- 각 요소를 구분하기 위해 번호를 붙임 => 인덱스(index)
- 인덱스 번호는 0부터 시작

❖ 값에 접근하기

- 인덱스 (Index)
  - 0부터 시작하는 각 값의 주소
  - 인덱스 사용하여 원하는 값 가져오거나 변경 가능
- 인덱싱
  - 인덱스로 리스트 값에 접근하는 것
- 리스트[접근할_인덱스]
  - 인덱스가 리스트 크기보다 크면 에러 발생

2. 리스트 생성, 값 변경, 값 추가

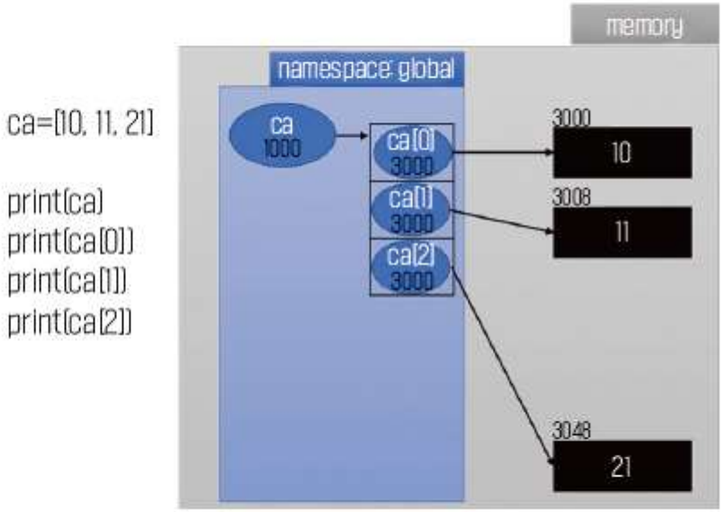
❖ 리스트 생성

- **규칙성 있는 변수명을 가진 변수 여러 개를 한 번에 자동 생성**
  - **ca 리스트로 정수값 10, 11, 21을 할당하면 ca[0], ca[1], ca[2]라는 인덱스(0, 1, 2)의 규칙성을 갖는 변수 3개를 자동 생성하고 3개의 변수를 묶은 변수 ca가 정의됨**

```
ca=[10,11,21]
print(ca) # ------> ①
print(ca[0]) # ----> ②
print(ca[1])
print(ca[2])
```

〈화면 출력〉

```
[10, 11, 21]
10
11
21
```



2. 리스트 생성, 값 변경, 값 추가

❖ 리스트 생성 예시

- lista 변수는 정수, 문자열, 실수를 묶어 정의
- 리스트 변수를 정의하면 인덱스 규칙성을 갖는 변수명 자동 생성

```
lista=[1, "python", 3.9, '프로그래밍']
print(lista)
print(lista[0])
print(lista[1])
print(lista[2])
print(lista[3])
```

①

〈화면 출력〉

```
[1, 'python', 3.9, '프로그래밍']
1
python
3.9
프로그래밍
```

2. 리스트 생성, 값 변경, 값 추가

❖ 값 변경하기

- 리스트 정의 후 리스트 요소를 대입 연산자를 사용하여 새로운 값으로 할당 가능

syntax : 리스트 값 변경

```
리스트명[인덱스] = 데이터 값
```

```
a=[1, 2, 3, 4, 5]
a[2]=30
print(a)
a[3]=40
print(a)
```

〈화면 출력〉

```
1, 2, 30, 4, 5]
[1, 2, 30, 40, 5]
```

2. 리스트 생성, 값 변경, 값 추가

❖ 값 추가하기

- append() 메소드
- 리스트명.append(추가할_값)
  - 리스트 이름 뒤에 마침표
  - append()의 소괄호 안에 추가할 값 삽입
- 리스트 정의 시 크기 결정
- 요소 개수에 따라 자동적으로 크기와 인덱스 결정
- 정해진 크기를 초과하는 요소 추가 시 사용

syntax : append() 메소드

```
리스트명.append(데이터 값)
```

2. 리스트 생성, 값 변경, 값 추가

❖ 리스트 값 추가 예제

- 빈 리스트 추가 : 값 할당 없이 리스트 정의 가능

```
listc=[]
print(listc)
listc.append(300)
print(listc)
listc.append("파이썬")
print(listc)
listc.append(3.7)
print(listc)
```

〈화면 출력〉

```
[]
[300]
[300, '파이썬']
[300, '파이썬', 3.7]
```

2. 리스트 생성, 값 변경, 값 추가

❖ 값 제거하기

- remove() 메소드
- 리스트.remove(제거할_값)
  - 리스트 이름 뒤에 마침표
  - remove()의 소괄호 안에 제거할 값 작성
  - 리스트 내 동일 값이 2개 이상 있는 경우, 첫 번째 발견된 요소만 삭제

  - 예) 리스트에서 특정 값을 제거하는 코드
    - subject = [‘국어’ , ‘수학’ , ‘영어’ , ‘국사’]
    - subject.remove(‘영어’)
    - print(subject)

    - 실행 결과
      - » [‘국어’ , ‘수학’ , ‘국사’]

2. 리스트 생성, 값 변경, 값 추가

❖ 값 제거하기

- **del 키워드**
- **del 리스트[제거할_인덱스]**

코드 3-6 리스트에서 값을 제거하는 코드

```
>>> clovers = ['클로버1', '클로버2', '클로버3']
>>> print(clovers[1])
클로버2
>>> del clovers[1]
>>> print(clovers)
['클로버1', '클로버3']
>>> print(clovers[1])
클로버3
```

del 키워드로 clovers 리스트의 1번 인덱스를 제거합니다.

2. 리스트 생성, 값 변경, 값 추가

❖ 특정 위치에 값 추가하기

- insert() 메소드
- 리스트.insert(인덱스, 추가할_값)
  - 리스트 이름 뒤에 마침표
  - insert()의 소괄호 안에 인덱스와 추가할 값 작성

❖ 여러 값 추가하기

- extend() 메소드
- 리스트.extend([값1, 값2, 값3 …])
  - 리스트 이름 뒤에 마침표
  - extend()의 소괄호 안에 추가할 값들의 리스트 작성

2. 리스트 생성, 값 변경, 값 추가

❖ 여러 값 가져오기

- **슬라이싱**
  - 여러 개의 값을 잘라서 가져오는 것
- **리스트[시작_인덱스:끝_인덱스+1]**
  - 원본 리스트는 변하지 않음

코드 3-8 리스트에서 여러 개의 값을 가져오는 코드

```
>>> week = ['월', '화', '수', '목', '금', '토', '일']
>>> print(week)
['월', '화', '수', '목', '금', '토', '일']
>>> print(week[2:5])
['수', '목', '금']
>>> print(week)
['월', '화', '수', '목', '금', '토', '일']
```

week 리스트의 2번부터 4번 인덱스를 가져옵니다.

3. 튜플

❖ 튜플 (Tuple)

- 여러 값을 함께 모을 수 있는 자료형
- 소괄호와 콤마 사용

• 사전적 의미는 Tuple과 List가 비슷

• 리스트는 '목록', 튜플은 'N개의 요소로 된 집합'

syntax : 튜플

```
변수명 = (요소1, 요소2, 요소3....)
```

- 리스트와 튜플 차이: 한 번 만든 값을 변경하고 싶지 않은 경우

List	Tuple
• 데이터 변경 가능(리스트 생성 후 추가/수정/삭제 가능) • 이름 그대로 목록 형식의 데이터를 다루는 데 적합	• 데이터 변경 불가능(튜플 생성 후 추가/수정/삭제 불가능)

3. 튜플

❖ 값 가져오기

- 대괄호 사용
- 튜플[접근할_인덱스]
  - 대괄호 안에 인덱스 넣어 해당 값 가져옴

코드 7-3 튜플에서 값을 가져오는 코드

```
>>> clovers = ('클로버1', '하트2', '클로버3')
>>> print(clovers[1])
하트2
```

00

4. 튜플 생성 및 특성

튜플 생성

- 값 할당 전 빈 튜플 생성 가능
- 서로 다른 자료형 포함 가능
- 괄호 생략 가능

코드 7-1 튜플을 만드는 코드

```
>>> my_tuple1 = ()
>>> print(my_tuple1)
()
>>> my_tuple2 = (1, -2, 3.14)
>>> print(my_tuple2)
(1, -2, 3.14)
>>> my_tuple3 = '앨리스', 10, 1.0, 1.2
>>> print(my_tuple3)
('앨리스', 10, 1.0, 1.2)
```

00

4. 튜플 생성 및 특성

▪ 튜플 생성

- 값을 한 개만 가진 튜플에서는 값 뒤에 쉼표 넣어줘야 함

코드 7-2 값이 한 개인 튜플을 만드는 코드

```
>>> my_int = (1)
>>> print(type(my_int))
<class 'int'>
>>> my_tuple = (1,)
>>> print(type(my_tuple))
<class 'tuple'>
```

4. 튜플 생성 및 특성

▪ 튜플 특성 활용 코드

- 데이터 생성 후 데이터 값이 변경되면 안되는 경우 유용하게 사용

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

- 튜플의 0번째 아이템은 할당을 통해 변경을 시도
- 오류 메시지를 통해 튜플의 변경 불가능성 확인

- 부득이 값 변경이 필요한 경우,
  리스트 형태로 item을 변경하여
  값 할당 가능

```
a=(1,2,3)
print(a, "a의 데이터형식은", type(a))
b=list(a)
print(b, "b의 데이터형식은", type(b))
b[0]=7
print(b)
```

〈화면 출력〉

```
(1, 2, 3) a의 데이터형식은 <class 'tuple'>
[1, 2, 3] b의 데이터형식은 <class 'list'>
[7, 2, 3]
```

5. 딕셔너리

❖ 딕셔너리 (Dictionary)

- 키(Key)와 값(Value)의 쌍으로 구성된 자료형
  - 키는 변경할 수 없는 자료형만
  - 값은 자료형 무관

- {키1: 값1, 키2: 값2, …}
  - 키와 값을 콜론(:)으로 묶음
  - 중괄호와 콤마 사용

syntax : 딕셔너리 생성

```
변수명 = {key1: value1, key2:value2, key3:value3,....}
```

00

5. 딕셔너리

딕셔너리 생성

- 값 할당 전 빈 딕셔너리 생성 가능
- 서로 다른 자료형 포함 가능
- key는 중복 사용 불가(고유한 값)

코드 7-7 딕셔너리를 만드는 코드

```
>>> my_dict1 = {}
>>> print(my_dict1)
{}
>>> my_dict2 = {0: 1, 1: -2, 2: 3.14}
>>> print(my_dict2)
{0: 1, 1: -2, 2: 3.14}
>>> my_dict3 = {'이름': '앨리스', '나이': 10, '시력': [1.0, 1.2]}
>>> print(my_dict3)
{'이름': '앨리스', '나이': 10, '시력': [1.0, 1.2]}
```

00

6. 딕셔너리 데이터 추가

❖ 키-값 추가하기

- 대괄호 사용
- 딕셔너리[추가할_키] = 추가할 값
  - 대괄호 안에 키 넣고 = 사용해 값 저장

코드 7-8 딕셔너리에 키-값을 추가하는 코드

```
>>> clover = {'나이': 27, '직업': '병사'}
>>> print(clover)
{'나이': 27, '직업': '병사'}
>>> clover['번호'] = 9
>>> print(clover)
{'나이': 27, '직업': '병사', '번호': 9}
```

6. 딕셔너리 데이터 추가

❖ 값에 접근하기

- 딕셔너리[접근할_키]
- 딕셔너리.get(접근할_키)
  - 키로 값에 접근
  - 딕셔너리에 없는 키 사용할 경우 get() 메소드 활용

코드 7-9 딕셔너리의 값에 접근하는 코드

```
>>> clover = {'나이': 27, '직업': '병사', '번호': 9}
>>> print(clover['번호'])
9
>>> clover['번호'] = 6
>>> print(clover['번호'])
6
>>> print(clover.get('번호'))
6
```

6. 딕셔너리 데이터 추가

❖ 딕셔너리 메서드

▪ 딕셔너리.items()

- 딕셔너리의 모든 아이템(키와 값)을 반환
  - items : 키와 값을 통칭하는 용어

▪ 딕셔너리.keys()

- 딕셔너리의 모든 키 반환

▪ 딕셔너리.values()

- 딕셔너리의 모든 값 반환

00

7. 연습문제

❖ 아래와 같은 출력 결과가 나오도록 빈칸을 채워보세요

```
>>> nums = [          ]
>>> print(nums)
[1, 2, 3]
```

❖ 다음 코드의 출력 결과를 적어보세요

```
>>> fruits = []
>>> print(fruits)
          
>>> fruits.append('자몽')
>>> print(fruits)
          
>>> fruits.append('멜론')
>>> print(fruits)
          
>>> fruits.append('레몬')
>>> print(fruits)
          
```

7. 연습문제

❖ 다음 코드의 출력 결과를 적어보세요

```
>>> fruits = ['자몽', '멜론', '레몬']
>>> print(fruits[1])
```

❖ 아래와 같은 출력 결과가 나오도록 빈칸을 채워보세요

```
>>> fruits = ['자몽', '멜론', '레몬']
>>> print(fruits)
['자몽', '멜론', '레몬']
>>>
>>> print(fruits)
['멜론', '레몬']
```

00

7. 연습문제

❖ 아래와 같은 출력 결과가 나오도록 빈칸을 채워보세요

```
>>> nums = (          )
>>> print(nums)
(1, 2, 3)
```

❖ 다음 코드의 출력 결과를 적어보세요

```
01:  my_tuple = (3.14, 2.71)
02:  print(my_tuple)
03:  print(my_tuple[1])
```

```
↳
```

00

7. 연습문제

❖ 다음 중 출력 결과가 다른 코드를 고르세요

①

```
>>> my_var = 1,
>>> print(type(my_var))
```

②

```
>>> my_var = (1,)
>>> print(type(my_var))
```

③

```
>>> my_var = (1)
>>> print(type(my_var))
```

④

```
>>> my_var = 1, 2
>>> print(type(my_var))
```

❖ 앨리스의 나이를 14로 바꾸는 코드를 작성하세요

```
01: alice = {'성별': '여', '나이': 13, '혈액형': 'AB'}
02: 
03: print(alice['나이'])
```

```
↳ 14
```

Notice

❖ 과제

- 1. 리스트 코드 3개 이상 작성하기
- 2. 리스트에 내용 추가/삭제하기
- 3. 튜플 코드 3개 이상 작성하기
- 4. 튜플 자료형을 리스트 자료형으로 변경하기
- 5. 딕셔너리 코드 3개 이상 작성하기
- 6. 딕셔너리에 데이터 추가하고 출력하기

❖ 다음 수업 내용

- 반복문
  - for문, range, continue, break, while, 이중 for문