

5주

반복문



염 희 균

목차

- 리스트와 반복문
- 딕셔너리와 반복문
- for 반복문
- 키워드로 정리하는 핵심 포인트
- 확인문제

시작하기 전에

[핵심 키워드] 리스트, 딕셔너리 반복문, 범위(range())

[핵심 포인트1] 여러 개의 값을 나타낼 수 있게 해주는 리스트, 딕셔너리 등의 자료형도 존재한다. 이번 절에서는 리스트와 딕셔너리 자료형에서 자료가 반복문에 의해 어떻게 활용되는지 살펴본다.

[핵심 포인트2] 특정 횟수 / 특정 시간만큼, 그리고 어떤 조건이 될 때까지 반복하는 등의 경우에 대해 알아본다.

시작하기 전에1

- 리스트 (list)
 - 여러 가지 자료를 저장할 수 있는 자료
 - 자료들을 모아서 사용할 수 있게 해 줌
 - 대괄호 내부에 자료들 넣어 선언

```
>>> array = [273, 32, 103, "문자열", True, False]
>>> print(array)
[273, 32, 103, '문자열', True, False]
```

시작하기 전에2

- **딕셔너리** (dictionary)
 - 키를 기반으로 값을 저장하는 것

```
{  
  키 ↓   값 ↓  
  "키A": 10,      # 문자열을 키로 사용하기  
  "키B": 20,  
  "키C": 30,  
  1:    40,      # 숫자를 키로 사용하기  
  False: 50     # 불을 키로 사용하기  
}
```

자료형	의미	가리키는 위치	선언 형식
리스트	인덱스를 기반으로 값을 저장	인덱스	변수 = []
딕셔너리	키를 기반으로 값을 저장	키	변수 = {}

반복문

- 프로그램에서 반복 업무를 진행하기 위한 대표적인 구문이다.

1. 컨테이너 자료형(컬렉션)을 이용한 반복문

- 리스트(List)
- 튜플(Tuple)
- 딕셔너리(Dictionary)

2. range() 함수를 이용한 반복문

- range(시작, 끝, 단계)
- Ex) range(1,10,1): 구구단

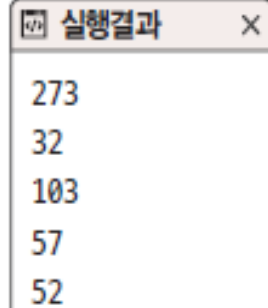
```
... 컬렉션
... for i in score: 콜론(:) 사용
...
... 들여쓰기: sum+=i
...
... 일반적인 for 명령의 구조
```

for 반복문 : 리스트와 함께 사용하기

- 문자열, 리스트와 조합하여 for 반복문을 사용

```
for 반복자 in 반복할 수 있는 것:  
    코드
```

```
01  # 리스트를 선언합니다.  
02  array = [273, 32, 103, 57, 52]  
03  
04  # 리스트에 반복문을 적용합니다.  
05  for element in array:  
06      # 출력합니다.  
07      print(element)
```



실행결과

```
273  
32  
103  
57  
52
```

for 반복문 : 리스트와 함께 사용하기

- 리스트와 for 반복문 예시

```
list = ['a', 'hello', 123, 3.14]
```

```
for i in list:
```

```
    print(i)
```

```
print('end')
```


딕셔너리 내부에 키가 있는지 확인하기

- get() 함수
 - 딕셔너리의 키로 값을 추출
 - 존재하지 않는 키에 접근할 경우 **None** 출력

```
01 # 딕셔너리를 선언합니다.
02 dictionary = {
03     "name": "7D 건조 망고",
04     "type": "당절임",
05     "ingredient": ["망고", "설탕", "메타중아황산나트륨", "치자황색소"],
06     "origin": "필리핀"
07 }
08
09 # 존재하지 않는 키에 접근해 봅니다.
10 value = dictionary.get("존재하지 않는 키")
11 print("값:", value)
12
13 # None 확인 방법
14 if value == None: → None과 같은지 확인만 하면 됩니다.
15     print("존재하지 않는 키에 접근했었습니다.")
```

실행결과

값: None
존재하지 않는 키에 접근했었습니다.

for 반복문 : 딕셔너리와 함께 사용하기

- for 반복문과 딕셔너리의 조합

```
for 키 변수 in 딕셔너리:  
    코드
```

for 반복문 : 딕셔너리와 함께 사용하기

```
01 # 딕셔너리를 선언합니다.
02 dictionary = {
03     "name": "7D 건조 망고",
04     "type": "당절임",
05     "ingredient": ["망고", "설탕", "메타중아황산나트륨", "치자황색소"],
06     "origin": "필리핀"
07 }
08
09 # for 반복문을 사용합니다.
10 for key in dictionary:
11     # 출력합니다.
12     print(key, ":", dictionary[key])
```

실행결과

```
name : 7D 건조 망고
type : 당절임
ingredient : ['망고', '설탕', '메타중아황산나트륨', '치자황색소']
origin : 필리핀
```

확인문제1

- list_a = [0, 1, 2, 3, 4, 5, 6, 7] 입니다. 다음 표의 함수들을 실행했을 때 list_a의 결과가 어떻게 나오는지 적어보세요

함수	list_a의 값
list_a.extend(list_a)	
list_a.append(10)	
list_a.insert(3, 0)	
list_a.remove(3)	
list_a.pop(3)	
list_a.clear()	

확인문제2

- 다음 반복문 내부에 if 조건문의 조건식을 채워서 100 이상의 숫자만 출력하게 만들어보세요.

```
numbers = [273, 103, 5, 32, 65, 9, 72, 800, 99]

for number in numbers:
    if :
        print("- 100 이상의 수:", number)
```

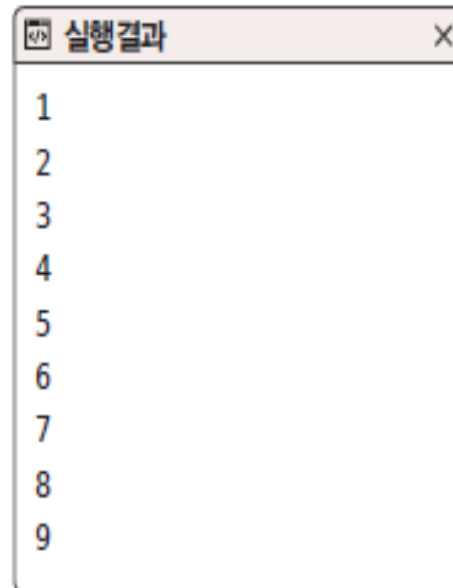
실행결과

- 100 이상의 수: 273
- 100 이상의 수: 103
- 100 이상의 수: 800

확인문제3

- 다음 빈칸을 채워서 실행결과처럼 숫자를 하나하나 모두 출력해보세요

```
list_of_list = [  
    [1, 2, 3],  
    [4, 5, 6, 7],  
    [8, 9],  
]
```



실행결과

1
2
3
4
5
6
7
8
9

확인문제4

- 다음 표에서 dict_a의 결과가 나오도록 빈칸을 채워 보세요.

dict_a의 값	dict_a에 적용할 코드	dict_a의 결과
<code>{}</code>	<input type="text"/>	<code>{"name": "구름"}</code>
<code>{"name": "구름"}</code>	<input type="text"/>	<code>{}</code>

확인문제5

- 다음 빈칸을 채워서 numbers 내부에 들어 있는 숫자가 몇 번 등장하는지를 출력하는 코드를 작성해 보세요.

숫자는 무작위로 입력해도 상관 없습니다.

```
numbers = [1,2,6,8,4,3,2,1,9,5,4,9,7,2,1,3,5,4,8,9,7,2,3]
```

```
counter = {}
```

```
for number in numbers:
```

```
    # 빈칸에 코드를 작성하세요
```

최종 출력

```
print(counter)
```

실행결과

```
{1: 3, 2: 4, 6: 1, 8: 2, 4: 3, 3: 3, 9: 3, 5: 2, 7: 2}
```


확인문제6

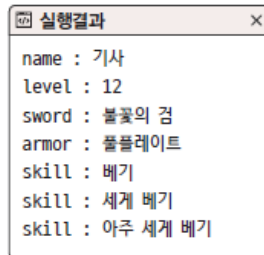
- 다음과 같은 방법으로 특정 값이 어떤 자료형 인지 확인 할 수 있다.

```
type("문자열") is str # 문자열인지 확인
type([]) is list      # 리스트인지 확인
type({}) is dict      # 딕셔너리인지 확인
```

- 예시를 참조해 다음 빈칸을 채워 실행 결과와 같이 출력 되게 만들어 보세요.

```
# 딕셔너리를 선언합니다.
character = {
    "name": "기사",
    "level": 12,
    "items": {
        "sword": "불꽃의 검",
        "armor": "풀플레이트"
    },
    "skill": ["베기", "세게 베기", "아주 세게 베기"]
}
```

```
# for 반복문을 사용합니다.
for key in character:
```



```
실행결과
name : 기사
level : 12
sword : 불꽃의 검
armor : 풀플레이트
skill : 베기
skill : 세게 베기
skill : 아주 세게 베기
```

range()함수를 이용한 반복문

- 범위 (range)
 - 특정 횟수만큼 반복해서 돌리고 싶을 때 for 반복문과 조합하여 사용

범위

- 예시
 - 매개변수에 숫자 한 개 넣은 범위

```
>>> a = range(5)
```

```
>>> a  
range(0, 5)
```

```
>>> list(range(10))  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

범위

- 매개변수에 숫자 두 개 넣은 범위

```
>>> list(range(0, 5)) → 0부터 (5-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 1, 2, 3, 4]
```

```
>>> list(range(5, 10)) → 5부터 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[5, 6, 7, 8, 9]
```

- 매개변수에 숫자 세 개 넣은 범위

```
>>> list(range(0, 10, 2)) → 0부터 2씩 증가하면서 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 2, 4, 6, 8]
```

```
>>> list(range(0, 10, 3)) → 0부터 3씩 증가하면서 (10-1)까지의 정수로 범위를 만듭니다.
```

```
[0, 3, 6, 9]
```

범위

- 범위 만들 때 매개변수 내부에 수식 사용하는 경우
 - 코드 특정 부분의 강조

```
>>> a = range(0, 10 + 1)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- 예시 - 나누기 연산자 사용

```
>>> n = 10
>>> a = range(0, n / 2) → 매개변수로 나눗셈을 사용한 경우 오류가 발생합니다.
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
```

- TypeError 발생

범위

- 정수 나누기 연산자

```
>>> a = range(0, int(n / 2)) → 실수를 정수로 바꾸는 방법보다
```

```
>>> list(a)
```

```
[0, 1, 2, 3, 4]
```

```
>>> a = range(0, n // 2) → 정수 나누기 연산자를 많이 사용합니다!
```

```
>>> list(a)
```

```
[0, 1, 2, 3, 4]
```

범위 range()와 반복문

`for i in range(1, 11, 3):` 시작 값 끝 값 증가 값
`print("*"*i)` ✓ 끝 값은 포함되지 않으므로 주의!
→ range(1,10) = 1에서 9까지 반복

✓ 증가 값을 생략하는 경우, default 값 1

```
for i in range(1,5):  
    print(i, ". 안녕하세요")
```

```
>>> 1. 안녕하세요  
     2. 안녕하세요  
     3. 안녕하세요  
     4. 안녕하세요
```

✓ 끝 값을 생략 불가

✓ 시작 값을 생략하는 경우, default 값 0

```
for i in range(5):  
    print("안녕하세요")
```

```
>>> 안녕하세요  
     안녕하세요  
     안녕하세요  
     안녕하세요  
     안녕하세요
```

```
for i in range(1, 10, 1) : #6단  
    print('{0} * {1} = {2}'.format(6,1,(6*i)))
```

for 반복문: 범위와 함께 사용하기

- for 반복문과 범위의 조합

for 숫자 변수 in 범위:

코드

```
01  # for 반복문과 범위를 함께 조합해서 사용합니다.
02  for i in range(5):
03      print(str(i) + "= 반복 변수")
04  print()
05
06  for i in range(5, 10):
07      print(str(i) + "= 반복 변수")
08  print()
09
10  for i in range(0, 10, 3):
11      print(str(i) + "= 반복 변수")
12  print()
```

실행결과

```
0 = 반복 변수
1 = 반복 변수
2 = 반복 변수
3 = 반복 변수
4 = 반복 변수

5 = 반복 변수
6 = 반복 변수
7 = 반복 변수
8 = 반복 변수
9 = 반복 변수

0 = 반복 변수
3 = 반복 변수
6 = 반복 변수
9 = 반복 변수
```


for 반복문 : 리스트와 범위 조합하기

- 몇 번 반복인지를 알아야 하는 경우

```
# 리스트를 선언합니다.  
array = [273, 32, 103, 57, 52]  
  
# 리스트에 반복문을 적용합니다.  
for element in array:  
    # 출력합니다.  
    print(element)
```

↓
현재 무엇을 출력하고 있는지 보다, 몇 번째 출력인지를 알아야 하는 경우가 있습니다.

```
01 # 리스트를 선언합니다.  
02 array = [273, 32, 103, 57, 52]  
03  
04 # 리스트에 반복문을 적용합니다.  
05 for i in range(len(array)):  
06     # 출력합니다.  
07     print("{}번째 반복: {}".format(i, array[i]))
```

실행결과	
0번째 반복:	273
1번째 반복:	32
2번째 반복:	103
3번째 반복:	57
4번째 반복:	52

for 반복문: 반대로 반복하기

- 역반복문

- 큰 숫자에서 작은 숫자로 반복문 적용
- `range()` 함수의 매개변수 세 개 사용하는 방법

```
01  # 역반복문
02  for i in range(4, 0 - 1, -1):
03      # 출력합니다.
04      print("현재 반복 변수: {}".format(i))
```

실행결과

현재 반복 변수: 4
현재 반복 변수: 3
현재 반복 변수: 2
현재 반복 변수: 1
현재 반복 변수: 0

for 반복문: 반대로 반복하기

- `reversed()` 함수 사용하는 방법

```
01  # 역반복문
02  for i in reversed(range(5)):
03      # 출력합니다.
04      print("현재 반복 변수: {}".format(i))
```

실행결과

현재 반복 변수: 4
현재 반복 변수: 3
현재 반복 변수: 2
현재 반복 변수: 1
현재 반복 변수: 0

확인문제

- 다음 표를 채워 보세요.

코드가 여러 개 나올 수 있는 경우 가장 간단한 형태를 넣어 주세요.

코드	나타내는 값
<code>range(5)</code>	<code>[0, 1, 2, 3, 4]</code>
<code>range(4, 6)</code>	
<code>range(7, 0, -1)</code>	
<code>range(3, 8)</code>	<code>[3, 4, 5, 6, 7]</code>
	<code>[3, 6, 9]</code>

키워드로 정리하는 핵심 포인트

- **리스트** : 여러 가지 자료를 저장할 수 있는 자료형
- **딕셔너리** : 키를 기반으로 여러 자료 저장하는 자료형
- **for 반복문** : 특정 코드를 반복해서 실행할 때 사용하는 기본 구문
- **범위** : 정수의 범위 나타내는 값으로, range() 함수로 생성