3주차

3-1. 리스트 자료형

리스트(list)

- 여러가지 자료를 저장할 수 있는 자료형
- 자료 묶음(목록)

```
array = [273, 32, 103, "Hello", True, -3.8]
print(array)
```

리스트 선언하고 요소에 접근하기

- 리스트를 생성하는 방법은 대괄호[]에 자료를 쉼표로 구분해서 입력
- 대괄호[] 내부에 넣는 자료를 요소(element)라고 함
- `[요소, 요소, 요소...]`
- 요소를 사용하려면 리스트 이름 바로 뒤에 대괄호[]를 입력하고, 자료의 위치를 나타내는 숫자를 입력
- 이 숫자를 인덱스라고 함
- 파이썬의 인덱스는 0부터 시작

```
array = [273, 32, 103, "Hello", True, -3.8]
print(array[0])
print(array[1])
print(array[2])
print(array[3])
print(array[4])
print(array[5])
```

- 인덱스 범위를 초과하는 요소에 접근하면? → IndexError: out of range

특정 인덱스의 값을 접근 및 변경

```
array = [273, 32, 103, "Hello", True, -3.8]
# 0번째 요소에 접근 및 변경
print(array[0])
array[0] = "변경"
print(array[0])
print()
# 맨 마지막 요소에 접근 및 변경
print(array[-1])
array[-1] = "변경2"
print(array[-1])
print()
# 이중으로 리스트 접근 연산자를 사용
print(array[3])
print(array[3][0])
print()

# 리스트 안에 리스트
list_a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(list_a)
print(list_a[0][2])
print(list_a[1][1])
```

리스트 연산자

```
# 리스트 선언
list_a = [1, 2, 3]
list_b = [4, 5, 6]

# 출력
print(list_a + list_b)
print(list_a * 3)
print()

# len함수 사용
print('list_a의 길이는', len(list_a))
```

리스트에 요소 추가하기: `append()` , `insert()`

```
# 리스트 선언
list_a = [1, 2, 3]
print(list_a)
print()

# 리스트 뒤에 요소 추가하기
list_a.append(4)
list_a.append(5)
print(list_a)
print()

# 리스트 중간에 요소 추가하기
list_a.insert(0, 10)
print(list_a)
```

- `append(element)` : 맨 뒤에 요소 추가
- `insert(index, element)` : `index` 위치에 요소 추가

리스트 요소 제거하기

1. 인덱스로 제거하기

```
# 리스트 선언
list_a = [0, 1, 2, 3, 4, 5]
print(list_a)
print()

# 제거 방법[1] - del
del list_a[1]
print(list_a)
print()

# 제거 방법[2] - pop()
list_a.pop(2)
print(list_a)
print()

# pop()을 하면? 자동으로 -1로 취급하여 맨 뒤의 요소 제거
```

```
list_a.pop()
print(list_a)
```

2. 값으로 제거하기

```
# 리스트 선언
list_a = [1, 2, 4, 2, 3, 7, 8]
print(list_a)
print()

list_a.remove(2) # 가장 앞의 2 하나만 제거
print(list_a)
print()
```

3. 모두 제거하기

```
# 리스트 선언
list_a = [1, 2, 4, 2, 3, 7, 8]
print(list_a)
print()

list_a.clear()
print(list_a)
```

리스트 내부에 있는지 확인하기: `in`

```
# 리스트 선언
list_a = [5, 7, 6, 1, 5, 9, 1, 3]
print(list_a)
print()

# 해당 값이 리스트 내부에 존재하는지 확인
print(27 in list_a)
print(3 in list_a)
print()
print(27 not in list_a)
print(3 not in list_a)
```

리스트의 요소 개수 확인하기: `count()`

```
# 리스트 선언
list_a = [1, 2, 4, 2, 3, 7, 8]
print(list_a)
print()

# 2의 개수?
print(list_a.count(2))
```

3-2. `for` 반복문

반복문

- 컴퓨터에 반복을 지시하는 방법
- 같은 코드를 계속 붙여 넣는 방법보다 더욱 편리

```
# 아래 출력문을 100번 반보하여 실행
for i in range(100):
    print("출력")
```

```
# 10번 반복하여 출력
for i in range(10):
    print('나무를', i, '번 찍었습니다.')
print('나무가 넘어갑니다.')
```

리스트와 함께 사용하기

- `for {반복자} in {반복할 수 있는 것}:`
- `{반복할 수 있는 것}` 에는 문자열, 리스트, 딕셔너리, 범위 등이 있음

```
# 리스트를 선언
array = [273, 32, 103, 57, 52]

# 리스트에 반복문을 적용
# 리스트의 요소 하나하나가 차례로 element라는 변수에 들어가며 반복
```

```
for element in array:
    print(element)
```

3-3. 딕셔너리 자료형

딕셔너리(Dictionary)

- '키'를 기반으로 '값'을 저장(key-value)
- 중괄호{}로 선언하며, '키: 값' 형태를 쉼표로 연결해서 생성
- 키는 문자열, 숫자, 불 등으로 선언 가능하지만, 일반적으로는 문자열 사용

```
# 딕셔너리 dict_a 선언
dict_a = {
    "name": "어벤저스 엔드게임",
    "type": "히어로 무비"
}
```

딕셔너리 요소에 접근하기

- 접근 시에는 []안에 키 값을 입력

```
# 딕셔너리 dict_a 선언
dict_a = {
    "name": "어벤저스 엔드게임",
    "type": "히어로 무비"
}
# dict_a 출력해보면?
print(dict_a)

# name 키 값에 접근
print(dict_a['name'])

# type 키 값에 접근
print(dict_a['type'])
```

```
# 딕셔너리 생성
dictionary = {
```

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

# 출력
print("name:", dictionary["name"])
print("type:", dictionary["type"])
print("ingredient:", dictionary["ingredient"])
print("origin:", dictionary["origin"])
print()

# 값 변경
dictionary["name"] = "8D 건조 망고"
print("name:", dictionary["name"])
```

딕셔너리에 값 추가하기/제거하기

- `딕셔너리[새로운 키] = 새로운 값`

```
# 위의 예시에서 이어서
# price 키와 그 값을 추가
dictionary["price"] = 5000
print(dictionary)
```

- 제거는 `del` 키워드 사용

```
# 위의 예시에서 이어서
del dictionary["ingredient"]
print(dictionary)
```

딕셔너리 내부에 키가 있는지 확인하기

- `in` 키워드

```
# 딕셔너리 생성
dictionary = {
```

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

# 사용자로부터 입력을 받음
key = input("> 접근하고자 하는 키: ")

# 출력
if key in dictionary:
    print(dicationary[key])
else:
    print("존재하지 않는 키에 접근하고 있습니다.")
```

- `get()` 함수
    - 존재하지 않는 키에 접근 시 `None` 반환

```
# 딕셔너리 생성
dictionary = {
    "name": "7D 건조 망고",
    "type": "당절임",
    "ingredient": ["망고", "설탕", "메타중아황산나트륨", "치자황색소"],
    "origin": "필리핀"
}

# 존재하지 않는 키에 접근 시도
value = dictionary.get("존재하지 않는 키")
print("값:", value)

# None 확인 방법
if value == None:
    print("존재하지 않는 키에 접근했었습니다.")
```

`for` 반복문: 딕셔너리와 함께 사용하기

- `for {키 변수} in {딕셔너리}`

```
# 딕셔너리 생성
dictionary = {
    "name": "7D 건조 망고",
    "type": "당절임",
    "ingredient": ["망고", "설탕", "메타중아황산나트륨", "치자황색소"],
    "origin": "필리핀"
}

# for 반복문 사용
for key in dictionary:
    print(key, ":", dictionary[key])
```

3-4. 범위 `range` 자료형

범위, `range()`

- 보통 `for` 문과 조합하여 사용

사용법

1. 매개변수에 숫자를 한 개 넣기
    - `range(A)` : 0~A-1 까지의 정수로 범위를 만듦
2. 매개변수에 숫자를 두 개 넣기
    - `range(A, B)` : A~B-1 까지의 정수로 범위를 만듦
3. 매개변수에 숫자를 세 개 넣기
    - `range(A, B, C)` : A~B-1까지의 정수로 범위를 만드는데, C만큼의 차이를 가짐

```
# 범위(range) 생성 및 출력
a = range(5)
print(a)
# list()함수로 리스트로 변경하여 출력
print(list(range(5)))

# 매개변수에 숫자를 두 개 넣기
print(list(range(1, 5)))
```

```
# 매개변수에 숫자를 세 개 넣기
print(list(range(1, 11, 3)))
```

- 주의점

```
n = 10
a = range(0, n / 2) # 매개변수로 나눗셈을 사용한 경우 오류 발생
# 매개변수로는 반드시 정수를 입력할 것!

# 다음과 같이 해결
a = range(0, int(n / 2))
a = range(0, n // 2)
```

for 반복문: 범위와 함께 사용하기

- `for {숫자 변수} in {범위}:`

```
for i in range(5):
    print(str(i) + "=반복 변수")
print()

for i in range(5, 10):
    print(str(i) + "=반복 변수")
print()

for i in range(0, 10, 3):
    print(str(i) + "=반복 변수")
print()
```

for 반복문: 리스트와 범위 조합하기

```
# 리스트 선언
array = [273, 32, 103, 57, 52]

# 리스트에 반복문을 적용
for element in array:
    print(element)
print()
```

```
# 리스트와 반복문을 모두 조합하여 사용하기
for i in range(len(array)):
    print(array[i])
```

for 반복문: 반대로 반복하기

```
# 역반복문
for i in range(4, 0, -1):
    print("반복변수:", i)
```

```
# 역 반복문: reversed()
for i in reversed(range(5)):
    print("반복변수:", i)
```

3-5. `while` 반복문

`while` 반복문

- `for` : 특정 횟수만큼 반복하는 경우 좋음
- `while` : 특정 조건에 따라 반복하는 경우 좋음
- `while {불 표현식}:`

```
# while 반복문을 사용합니다.
while True:
    print(".", end="")
```

- {불 표현식}이 `True` 인 경우에만 반복
    - `False` 이면 반복 자체를 안 함

`while` 반복문: `for` 반복문처럼 사용하기

```
# 반복 변수를 기반으로 반복하기
i = 0
while i < 10:
```

```
    print(i, "번째 반복입니다.")
    i += 1
```

while 반복문: 상태를 기반으로 반복하기

```
# 변수 선언
list_test = [1, 2, 1, 2]
value = 2

# list_test 내부에 value가 있다면 반복
while value in list_test:
    list_test.remove(value)

# 결과 출력
print(list_test)
```

while 반복문: break 키워드, continue 키워드

break

- 반복문을 벗어날 때 사용

```
# 반복 변수 생성
i = 0

# 무한 반복
while True:
    print(i, "번째 반복문입니다.")
    i = i + 1
    # 반복을 종료 질의
    input_text = input("> 종료하시겠습니까?(y / n) : ")
    if input_text in ["y", "Y"]:
        print("반복을 종료합니다.")
        break
```

continue

- 현재의 반복을 종료하고 다음 반복으로 넘어감

```
# 10 이상인 수만 리스트에서 출력하는 코드
# 변수를 선언
numbers = [5, 15, 6, 20, 7, 25]

# 반복문을 돌립니다.
for number in numbers:
    if number < 10:
        continue
    print(number)
```

```
# 똑같은 일을 하는 코드, 그러나 continue 사용하지 않음

# 변수를 선언
numbers = [5, 15, 6, 20, 7, 25]

# 반복문을 돌립니다.
for number in numbers:
    if number >= 10:
        print(number)
```

- 처음부터 조건을 걸고 싶을 때 좋음