세종사이버대학교

파이썬기초

2주차 - 3교시

파이썬의 자료형 - 2

학습내용

- 자료형의 구분 - 리스트, 튜플, 사전, 집합

학습목표

- 리스트, 튜플, 사전, 집합에 대해 이해하고 사용할 수 있다.

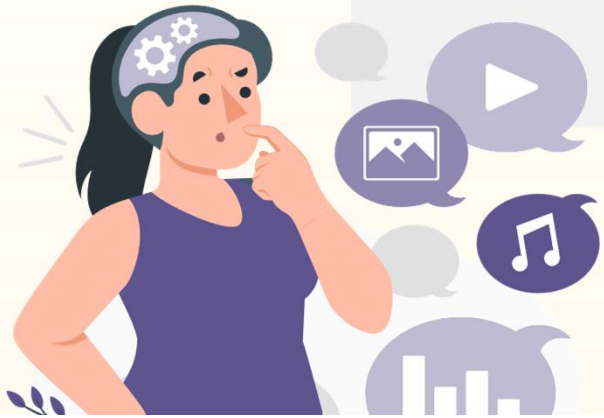
생각해 봅시다

프로그램 내부에서 많은 데이터를
저장하고 쉽게 처리할 수 있는
방법에 대해 생각해봅시다.

01

자료형의 구분

리스트, 튜플, 사전, 집합



1 | 리스트

01 리스트(List)

리스트

하나의 변수에 여러 값을 저장하는 자료형

- 하나씩 사용하던 박스(변수)를 **한 줄로 붙여 놓은 것**
- 박스(변수)를 한 줄로 붙인 후 전체에 **이름(list)**을 지정

1 | 리스트

01 리스트(List)

- 리스트의 요소 각각은 list[0], list[1], list[2], list[3] 처럼 **인덱스 번호**를 붙여 사용

1 | 리스트

02 리스트의 개념과 필요성

- 입력해서 사용해야 할 변수가 100개라면
  변수를 선언하고 할당하는 것이 굉장히 힘듦

리스트를 사용하면 보다 효과적으로 처리가 가능한 경우

› 학생 300명의 점수에 학점을 부여해야 하는 경우

1 | 리스트

03 리스트 선언 방법

- 대괄호 안에 **값**을 선언함

```
리스트명 = [값1, 값2, 값3, …]
```

1 | 리스트

04 리스트 생성 예제

```
In [2] : #리스트
         list = [10, 20, 30, 40]
         list
Out [2] : [10, 20, 30, 40]
In [3] : type(list)
Out [3] : list
In [4] : print(list)
         [10, 20, 30, 40]
```

1 | 리스트

05 인덱싱(Indexing)

인덱싱

리스트에 저장되어 있는 값에 접근하기 위해 이 값의 상대적인 주소(offset)를 사용하는 것

- **상대적인 주소**를 인덱스 값이라고 함
- 인덱스 값은 **0~ 리스트의 길이-1** 까지 범위를 가짐

- len( ) 함수는 리스트의 길이,
  즉 리스트 안에 있는 요소의 개수를 반환함

1 | 리스트

05 인덱싱(Indexing)

In [7] :
```
#리스트의 indexing
print(len(list)) #리스트의 길이 : 리스트가 가지는 요소의 개수
```
```
4
```

1 | 리스트

05 인덱싱(Indexing)

```
In [8] : print(list[0])
         print(list[1])
         print(list[2])
         print(list[3])
```

```
10
20
30
40
```

1 | 리스트

05 인덱싱(Indexing)

```
In [9] : print(list[4])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[9], line 1
----> 1 print(list[4])

IndexError: list index out of range
```

1 | 리스트

06 슬라이싱(Slicing)

- **리스트에서 파생된 기능** 중 하나
- 리스트의 인덱스 기능을 사용하여
  전체 리스트에서 일부를 잘라내어 사용
- 앞에서 선언한 리스트는 0~3 까지의 인덱스를 가진
  4개의 요소를 가지고 있음

```
값       [10, 20, 30, 40]
인덱스    0   1   2   3
```

1 | 리스트

06 슬라이싱(Slicing)

- 슬라이싱의 기본 문법

> 변수명[시작 인덱스 : 마지막 인덱스+1]

1 | 리스트

06 슬라이싱(Slicing)

```
In [10] : list = [10, 20, 30, 40]

In [12] : list[0:4]
Out [12] : [10, 20, 30, 40]

In [13] : list[:4]
Out [13] : [10, 20, 30, 40]

In [14] : list[0:]
Out [14] : [10, 20, 30, 40]
```

1 | 리스트

06 슬라이싱(Slicing)

```
In [15] : list[2:4]
Out [15] : [30, 40]

In [16] : list[2:]
Out [16] : [30, 40]
```

1 | 리스트

07 리버스 인덱스(Reverse Index)

- 마지막 값부터 -1을 할당하여 첫 번째 값까지 **역순**으로 진행하는 방식

```
In [18] : list[-4:]
Out [18] : [10, 20, 30, 40]
```

1 | 리스트

08 증가값(Step)

- 슬라이싱에서는 시작 인덱스와 마지막 인덱스 외에도 **마지막 자리에 증가값**을 사용

> 변수명[시작 인덱스:마지막 인덱스:증가값]

1 | 리스트

08 증가값(Step)

```
In [19] : # list : step
          list[0:4:2]
Out [19] : [10, 30]

In [20] : list[::2]
Out [20] : [10, 30]

In [21] : list[::-1]
Out [21] : [40, 30, 20, 10]
```

1 | 리스트

09 리스트 덧셈 연산

- **덧셈 연산**으로 두 리스트를 합치면,
  각각의 리스트가 하나의 리스트로 합쳐져서 출력됨

```
In [23] : #리스트의 덧셈
          list_a = [10, 20, 30, 40]
          list_b = [50, 60, 70, 80]
          list_total = list_a + list_b
          list_total

Out [23] : [10, 20, 30, 40, 50, 60, 70, 80]
```

1 | 리스트

10 리스트 곱셈 연산

- 리스트에 n을 곱했을 때
  해당 리스트를 n배만큼 늘려 줌

```
In [25] : #리스트의 곱셈 연산
          list_result = list_a * 3
          list_result

Out [25] : [10, 20, 30, 40, 10, 20, 30, 40, 10,
           20, 30, 40]
```

1 | 리스트

11 리스트 in 연산

- **포함 여부를 확인하는 연산**으로 하나의 값이 해당 리스트에 들어있는지 확인할 수 있음

In [28] :
```
#리스트의 in 연산
print(130 in list_a)
print(50 in list_b)
```

```
False
True
```

1 | 리스트

12 리스트 조작함수

함수	설명	사용법
append()	리스트 맨 뒤에 항목을 추가한다.	리스트명.append(값)
pop()	리스트 맨 뒤의 항목을 빼낸다. (리스트에서 해당 항목이 삭제된다.)	리스트명.pop()
sort()	리스트의 항목을 정렬한다.	리스트명.sort()
reverse()	리스트 항목의 순서를 역순으로 만든다.	리스트명.reverse()
index()	지정한 값을 찾아 해당 위치를 반환한다.	리스트명.index(찾을 값)
insert()	지정된 위치에 값을 삽입한다.	리스트명.insert(위치,값)

1 | 리스트

12 리스트 조작함수

함수	설명	사용법
remove()	리스트에서 지정한 값을 삭제한다. 단, 지정한 값이 여러 개면 첫 번째 값만 지운다.	리스트명.remove(지울 값)
extend()	리스트 뒤에 리스트를 추가한다. 리스트의 더하기(+)연산과 기능이 동일하다.	리스트명.extend(추가할 리스트)
count()	리스트에서 해당 값의 개수를 센다.	리스트명.count(찾을 값)
clear()	리스트의 내용을 모두 지운다.	리스트명.clear()
del()	리스트에서 해당 위치의 항목을 삭제한다.	del(리스트명[위치])
len()	리스트에 포함된 전체 항목의 개수를 센다.	len(리스트명)

1 | 리스트

12 리스트 조작함수

함수	설명	사용법
copy()	리스트의 내용을 새로운 리스트에 복사한다.	새리스트=리스트명.copy()
sorted()	리스트의 항목을 정렬해서 새로운 리스트에 대입한다.	새리스트=sorted(리스트)

1 | 리스트

12 리스트 조작함수

리스트 조작함수 예제

```
In [29] : list_numbers = [10, 20, 30, 40]

In [30] : list_numbers.append(100)
          list_numbers

Out [30] : [10, 20, 30, 40, 100]

In [31] : list_numbers.extend([1000, 2000])
          list_numbers

Out [31] : [10, 20, 30, 40, 100, 1000, 2000]
```

1 | 리스트

12 리스트 조작함수

리스트 조작함수 예제

```
In [32] : list_numbers.insert(0, 99)
          list_numbers
Out [32] : [99, 10, 20, 30, 40, 100, 1000, 2000]

In [33] : list_numbers.remove(99)
          list_numbers
Out [33] : [10, 20, 30, 40, 100, 1000, 2000]
```

1 | 리스트

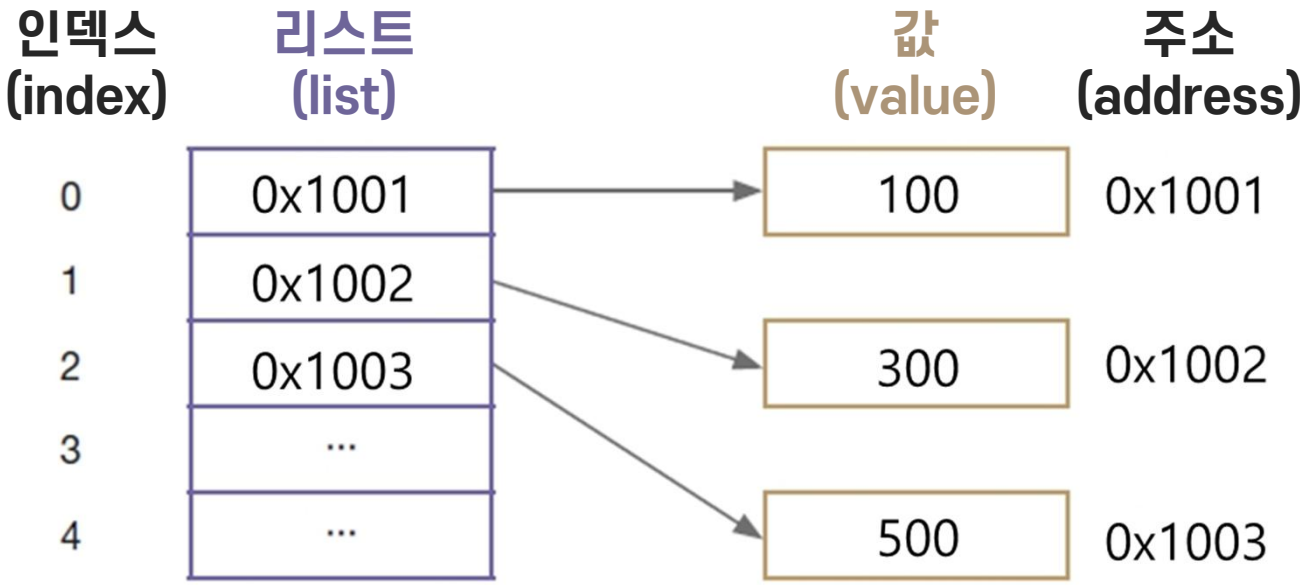
12 리스트 조작함수

리스트 조작함수 예제

```
In [34] : del list_numbers[4]
          list_numbers
Out [34] : [10, 20, 30, 40, 1000, 2000]
```

1 | 리스트

13 리스트의 메모리 구조



2 | 튜플

01 튜플

튜플

리스트와 같이 여러 타입의 데이터들을 순서에 따라 나열한 것

- **( ) 괄호** 내에 원소 값들을 콤마(,)로 구분해서 순차적으로 표현함

예 ('A', 'B', 'C', 'D', 'F')

2 | 튜플

01 튜플

- 리스트와는 달리 원소들의 추가, 삭제, 변경이 불가능함

```
In [35] : #튜플
          list_score = (100, 90, 60, 80, 70)
          print(list_score)
          print(type(list_score))
          (100, 90, 60, 80, 70)
          <class 'tuple'>
```

2 | 튜플

01 튜플

- 튜플 생성 시 소괄호 생략도 가능

```
In [36] : list_score_a = 100, 90, 60, 80, 70
          print(list_score_a)
          print(type(list_score_a))
          (100, 90, 60, 80, 70)
          <class 'tuple'>
```

2 | 튜플

02 튜플의 삭제

```
In [37]: del list_score_a

In [37]: list_score_a
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[38], line 1
----> 1 list_score_a

NameError: name 'list_score_a' is not defined
```

2 | 튜플

03 튜플의 활용

- 리스트처럼 **'튜플명[인덱스]'**를 사용
- 튜플도 리스트와 같이 인덱싱을 이용한 범위에 접근이 가능

튜플의 덧셈 및 곱셈 연산도 가능

3 | 집합

01 집합(Set)

집합

여러 유형의 데이터들을 순서에 관계없이 모아둔 것

- 수학의 **집합 개념**과 동일함
- **중괄호 { } 내부**에 원소들을 콤머로 구분해서 표현함

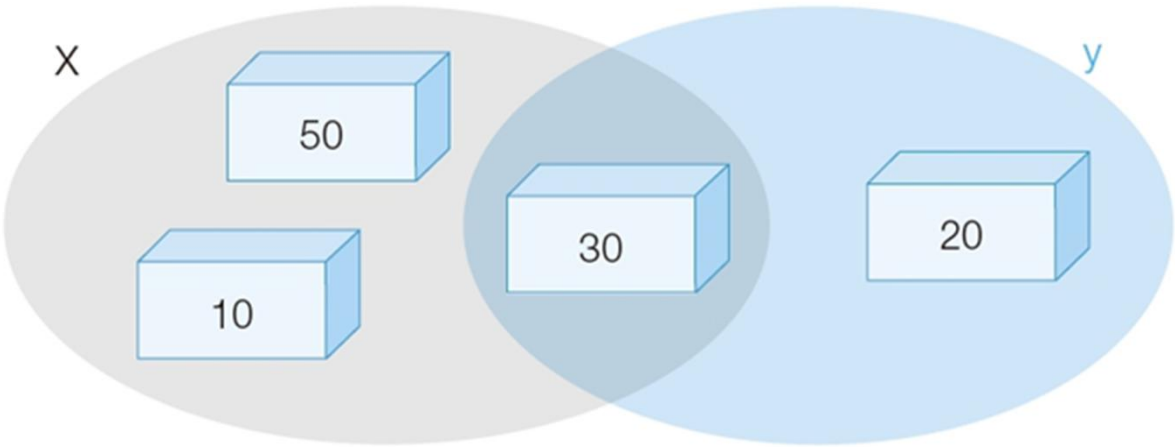
예 {10, 30, 50}

집합은 동일한 값을 중복 포함할 수 없고,
각 원소를 개별적으로 참조할 수 없음

3 | 집합

01 집합(Set)

- 수학의 집합연산(합집합, 교집합, 차집합 등) 지원함



x | y = {10, 20, 30, 50} x & y = {30}

3 | 집합

01 집합(Set)

- 수학의 집합연산(합집합, 교집합, 차집합 등) 지원함

```
In [45] : # set
          x = {10, 30, 50}
          y = {30, 20, 50, 40}
          print(x | y)     # 합집합
          print(x & y)     # 교집합
          print(x - y)     # 차집합
```

결과(Console)

```
{50, 20, 40, 10, 30}
{50, 30}
{10}
```

4 | 사전

01 딕셔너리(Dictionary)

딕셔너리

"키:값" 형태의 데이터들의 집합

- 중괄호 { } 내에 **"키:값"** 형태의 원소들을 콤마(,)로 구분해서 저장

예 {"name":"손흥민", "grade":1, "major":"축구"}

4 | 사전

01 딕셔너리(Dictionary)

키(key)

데이터 값을 구별하는 식별자

- 원소 값(데이터)을 참조할 때 **인덱스(식별 키)**로 사용함

  → "딕셔너리명[키]" 형식으로 원소 값 참조함

4 | 사전

01 딕셔너리(Dictionary)

- 딕셔너리는 여러 정보를 하나의 변수로 표현할 때 유용

student

name	grade	major
손흥민	4	축구

4 | 사전

01 딕셔너리(Dictionary)

- 딕셔너리는 여러 정보를 하나의 변수로 표현할 때 유용

In [46] :
```
# dict
student = {'name' : '손흥민', 'grade' : 4, 'major' : '축구'}
print(student)
print(student['name'], student['grade'], student['major'])
```

```
{'name' : '손흥민', 'grade' : 4, 'major' : '축구'}
손흥민 4 축구
```

4 | 사전

02 딕셔너리에 데이터 추가

- '딕셔너리명[키]=값' 형식으로 쌍을 추가
- 이미 존재하는 키를 사용하면 새로운 데이터가 추가되는 것이 아니라 기존 값이 변경됨

```
In [47] : student['age'] = 30
          student

Out [47] : {'name' : '손흥민', 'grade' : 4, 'major' : '축구', 'age' : 30}
```

4 | 사전

03 딕셔너리의 데이터 삭제

- del 딕셔너리명[키] 형식으로 삭제

```
In [48] : del student['age']
          student
Out [48] : {'name' : '손흥민', 'grade' : 4,  'major' : '축구'}
```

4 | 사전

04 딕셔너리의 활용

딕셔너리명.get(키) 함수
키로 값에 접근할 수 있음

✔ 딕셔너리명.keys()는 딕셔너리의 모든 키를 반환

딕셔너리명.items() 함수
키와 값의 쌍을 튜플 형태로도 구할 수 있음

딕셔너리명.values() 함수
딕셔너리의 모든 값을 리스트로 만들어 반환함

→ 해당 키가 있는지 없는지는 in을 사용해 확인

4 | 사전

04 딕셔너리의 활용

```
In [49] : student.get('name')
Out [49] : '손흥민'

In [50] : student.keys()
Out [50] : dict_keys(['name', 'grade', 'major'])

In [51] : student.items()
Out [51] : dict_items([('name', '손흥민'), ('grade', 4), ('major', '축구')])
```

4 | 사전

04 딕셔너리의 활용

```
In [52] : student.values()
Out [52] : dict_values(['손흥민', 4, '축구'])

In [53] : 'name' in student
Out [53] : True
```

5 | None

01 None

None

'자료값 없음'을 나타냄

- 변수를 정의할 때, 또는 프로그램의 실행 도중 변숫값을 None으로 지정함

```
In [55] : var_temp = None
          type(var_temp)
Out [55] : NoneType
```

Q1

메모리의 특정 위치에 이름을 붙여 프로그램이 실행되는 동안 사용하는 것을 무엇이라고 하는가?

1. 주석
2. 파이썬
3. 변수
4. 값

Q1

메모리의 특정 위치에 이름을 붙여 프로그램이
실행되는 동안 사용하는 것을 무엇이라고 하는가?

1. 주석
2. 파이썬
3. 변수
4. 값

정답

3번

해설

프로그램이 실행되는 동안 사용하기 위해 확보한 메모리의
특정 저장공간을 만들고 이름을 붙여 사용하는 것을 변수라 합니다.

Q2 다양한 데이터를 출력하기 위해 사용하는 함수는?

1. insert()
2. print()
3. input()
4. len()

Q2 다양한 데이터를 출력하기 위해 사용하는 함수는?

1. insert()
2. print()
3. input()
4. len()

정답 2번

해설 파이썬에서 제공하는 표준 출력 함수는 print() 함수입니다.

Q3 파이썬에서 자료형은 어느 시점에 정해지는가?

1. 프로그램 종료 시
2. 프로그램 시작 시
3. 프로그램 실행 시
4. 프로그램 컴파일 시

Q3 파이썬에서 자료형은 어느 시점에 정해지는가?

1. 프로그램 종료 시
2. 프로그램 시작 시
3. 프로그램 실행 시
4. 프로그램 컴파일 시

정답

3번

해설

파이썬은 동적 타이핑 언어로써 변수의 자료형은 프로그램이 실행되는 시점에 정해집니다.

Q4 문자열을 다루는 자료형이 무엇인지 고르시오.

1. str
2. bool
3. int
4. list

Q4 문자열을 다루는 자료형이 무엇인지 고르시오.

1. str
2. bool
3. int
4. list

정답 1번

해설 문자열을 저장하는 자료형은 str입니다.

Q5

집합 형태로 다양한 데이터를 저장하는 자료형이 아닌 것을 고르시오.

1. dict
2. tuple
3. list
4. None

Q5

집합 형태로 다양한 데이터를 저장하는 자료형이 아닌 것을 고르시오.

1. dict
2. tuple
3. list
4. None

정답

4번

해설

집합 형태로 데이터를 저장하는 타입은 리스트(list), 튜플(tuple), 집합(set), 딕셔너리(dictionary)가 있습니다.

변수의 이해

- **변수**는 데이터를 저장할 메모리공간을 생성하고 이름을 정해두어 데이터를 저장하고 필요한 시점에 이름을 이용해 메모리에 접근할 수 있도록 하는 것을 의미
- 변수 선언 시 **명명 규칙**에 맞게 선언해주어야 함
- 파이썬의 표준 입출력 함수는 **input( ), print( ) 함수**를 사용함

파이썬의 자료형

- 숫자를 다루는 자료형은 **정수(int), 실수(float)** 자료형
- 문자열을 다루는 자료형은 **문자열(str)**이고, 다양한 형태로 문자열을 다룰 수 있음
- 논리값을 저장하는 자료형은 **불린(Boolean)** 타입
- 집합 형태로 많은 데이터를 다룰 때는 **리스트(list), 튜플(tuple), 집합(set), 딕셔너리(dictionary)**를 사용함
- **list 타입**을 가장 많이 사용하고, 다양한 기능과 함수를 제공함

세종사이버대학교

파이썬기초

NEXT

파이썬의 연산자