4. 빅데이터 처리를 위한 파이썬 기초 03

- 데이터 다루기

목차

- 데이터 유형
- 시퀀스 자료형 이해하기
- 리스트 조작하기
- 딕셔너리 조작하기

데이터 유형

- 숫자형
- 문자열
- 논리형
- 리스트
- 딕셔너리
- 집합
- 튜플

시퀀스 자료형 이해하기

- 어떤 객체가 순서를 가지고 나열되어 있는 자료형

- 문자열 strdata = 'abcde'
- 리스트 listdata = [1, [2,3], 'hello']
- 튜플 tupledata = (10, 20, 30)

시퀀스 자료형 이해하기

- 시퀀스 자료형의 공통적 특성

특 성	설 명
인덱싱	인덱스를 통해 해당 값에 접근할 수 있음. 인덱스는 0부터 시작.
슬라이싱	특정 구간의 값을 취할 수 있음. 구간은 시작 인덱스와 끝 인덱스로 정의.
연결	'+'연산자로 두 시퀀스 자료를 연결하여 새로운 시퀀스 자료를 생성.
반복	'*' 연산자로 두 시퀀스 자료를 연결하여 새로운 시퀀스 자료를 생성.
멤버체크	'in'키워드로 특정 값이 시퀀스 자료의 요소인지 확인 가능.
크기정보	len()을 이용해 시퀀스 자료의 크기를 알 수 있음. 시퀀스 자료의 크기는 문자열의 경우는 문자의 개수, 리스트와 튜플은 멤버의 개수이다.

시퀀스 자료형 이해하기

- 인덱싱

코드

```
strdata = 'Hello Python!'
listdata = [1,2, [1, 2, 3]]

print(strdata[6])
print(strdata[-2])

print(listdata[0])
print(listdata[-1])
print(listdata[2][-1])
```

실행결과

strdata	H	e	l	l	o		P	y	t	h	o	n	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12
	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

시퀀스 자료형 이해하기

- 슬라이싱
  - [시작인덱스:끝인덱스:스텝]
  - 시작 이상 ~ 끝 미만
  - 스텝 생략 시 기본값 1.

코드

```
strdata = 'Hello Python!'

print(strdata[1:5])
print(strdata[:7])
print(strdata[9:])
print(strdata[:-3])
print(strdata[-3:])
print(strdata[:])
print(strdata[::2])
```

실행결과

strdata	H	e	l	l	o		P	y	t	h	o	n	!
index	0	1	2	3	4	5	6	7	8	9	10	11	12
	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

시퀀스 자료형 이해하기

- 연결
  - 합치기
  - 문자열 + 문자열
  - 리스트 + 리스트
  - 튜플 + 튜플
  - 리스트 + 튜플 (X)

코드

```
strdata1 = 'Hello '
strdata2 = 'Python!'
strdata3 = 'Friend!'
listdata1=[1,2,3]
listdata2=[4,5,6]

print(strdata1 + strdata2)
print(strdata1 + strdata3)
print(listdata1 + listdata2)
```

실행결과

시퀀스 자료형 이해하기

- 반복

코드

```
strdata1 = '니가  '
strdata2 = '너무~ '
strdata3 = ' 보고싶어!'

print(strdata1 + strdata2 * 2 + strdata3 )
```

실행결과

시퀀스 자료형 이해하기

- 크기정보
  - 모든 시퀀스 자료는 고정 길이
  - len( ) : 시퀀스 자료를 구성하는 요소의 개수 구하는 함수
  - 문자열 : 문자 개수
  - 리스트와 튜플 : 멤버의 개수

코드

```
strdata1 = 'Hello Python!'
strdata2 = '안녕 파이썬~~~'
listdata=['a', 'b', 'c',strdata1, strdata2]

print(len(strdata1))
print(len(strdata2))
print(len(listdata))
print(len(listdata[3]))
```

실행결과

시퀀스 자료형 이해하기

- 멤버체크
  - <값> in <자료>
  - 자료 안에 값이 있으면 True
    없으면 False

코드

```
listdata = [1,2,3,4]
chk1 = 5 in listdata
chk2 = 3 in listdata
print(chk1);  print(chk2)

strdata = 'Hello Python!'
chk3 = 'P' in strdata
chk4 = 'p' in strdata
print(chk3);  print(chk4)
```

실행결과

문자열 포맷팅 이해하기

- 변하는 값을 포함하는 문자열을 표현하기 위해 하나의 양식으로 문자열을 만드는 것.

포맷 문자열	설 명
%s	문자열에 대응
%c	문자나 기호 하나에 대응
%f	실수에 대응
%d	정수에 대응
%%	'%' 라는 기호 자체를 표시

문자열 포맷팅 이해하기

코드

```
print( "어떤 것이 더 중요할까요?(2가지 입력)")
str1 = input("첫번째 : ")
str2 = input("두번째 : ")
print('%s과 %s 중에 어떤 것이 더 중요할까?' %(str1, str2))

num1 = float(input("당신의 키는 ? "))
num2 = float(input("당신의 몸무게는 ?  "))
print('저의 키는%.2fcm이고, 몸무게는 %.2fkg 입니다.'  %(num1, num2))

print('프로젝트의 진행률은 %d%%입니다.' %num2)
```

실행결과

리스트(List) 조작하기

리스트 이해하기

- [ ] 로 생성 listdata=[1, '2', [2,3], {4,5,6}]
- list([ ]) 로 생성 listdata=list([1,2,3,4,5])
- 연속된 자료를 생성하기 위해 사용
- 순서 있음
- 인덱싱과 슬라이싱 가능.

리스트 이해하기

- 여러 가지 리스트의 유형
  - 숫자, 문자열, 리스트, 튜플, 딕셔너리 등을 포함.

코드

```
>>> a = [ ]    # a = list()
>>> b = [1, 2, 3]
>>> c = ['Life',  'is',  'too',  'short']
>>> d = [1, 2, 'Life',  'is']
>>> e = [1, 2, ['Life',  'is']]
```

리스트 이해하기

- 리스트 수정하기
  - 인덱스로 접근하여 수정

코드

```
 #리스트 수정 – 하나의 요소 수정
>>> b=[1,2,3]
>>> b[2] = 4
>>> b

#리스트 수정 – 여러개의 요소 수정
>>> b[1:3]
[2, 4]
>>> b[1:3]=['a', 'b', 'c']
>>> b
[1, 'a', 'b', 'c']
```

리스트 이해하기

- 리스트 요소 삭제하기

코드

```
#리스트의 요소 삭제
>>> b[1:3]=[]
>>> b
[1, 'c']

#del함수 이용하여 요소 삭제
>>> del b[1]
>>> b
[1]
```

리스트 관련 함수

- append() - 새로운 요소 추가(마지막에 추가)

코드

```
>>> num = list(range(5))
>>> num
[0, 1, 2, 3, 4]

>>> num.append(5)
>>> num.append(6)
>>> num
[0, 1, 2, 3, 4, 5, 6]
```

리스트 관련 함수

- sort() - 요소 정렬 하기
  - 원본 리스트를 정렬한 형태로 변경

코드

```
>>> namelist = ['이점숙', '김은경', '박순화']
>>> print('정렬 전 :', namelist)
정렬 전 : ['이점숙', '김은경', '박순화']

>>> namelist.sort()
>>> print('정렬 후 :', namelist)
정렬 후 : ['김은경', '박순화', '이점숙']
```

리스트 관련 함수

- reverse() – 역순으로 만들기
  - 원본 리스트가 변경 됨. 리스트 요소들을 순서대로 정렬한 다음 다시 역순으로 정렬하는 것이 아니라 그저 현재의 리스트를 그대로 거꾸로 뒤집을 뿐이다.

코드

```
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']
```

리스트 관련 함수

- index(x) - x라는 값이 있으면 x의 위치값을 리턴.
  0부터 시작.

코드

```
>>> a = ['b', 'c', 'a']

>>> a.index('a')
2
```

리스트 관련 함수

- insert (x, y) – 리스트의 x번째 위치에 y를 삽입하기

코드

```
>>> fruit =[ '배', '참외', '포도']

>>> fruit.insert(1, '수박')

>>> fruit
['배', '수박', '참외', '포도']
```

리스트 관련 함수

- remove(x) - 첫번째로 나오는 x 요소 제거하기

코드

```
>>> fruit
['배', '수박', '참외', '포도']

>>> fruit.append('수박')
>>> fruit
['배', '수박', '참외', '포도', '수박']

>>> fruit.remove('수박')
>>> fruit
['배', '참외', '포도', '수박']
```

리스트 관련 함수

- pop()
  - 마지막 요소 가져오면서(반환) 리스트에서 제거
- pop(x)
  - 리스트의 x번째 요소 가져오면서 리스트에서 삭제

코드

```
 >>> fruit
['배', '수박', '참외', '포도']

>>> fruit.pop()
 ' 포도'
>>> fruit
['배', '수박', '참외']

>>> fruit.pop(0)
'배'
>>> fruit
['수박', '참외']
```

리스트 관련 함수

- clear() – 전체 리스트 요소 제거하기

코드

```
>>> fruit
['참외', '포도']

>>> fruit.clear()

>>> fruit
[]
```

리스트 관련 함수

- max(리스트명)
- min(리스트명)
- sum(리스트명)

코드

```
>>> a=[61,82,43,65,88,43,78]

>>> max(a)
88

>>> min(a)
43

>>> sum(a)
460
```

리스트 관련 함수

- del - 특정 위치의 요소 제거하기
- del 리스트명 – 리스트 전체 제거

코드

```
>>> a=[61,82,43,65,88,43,78]

>>> del a[0]
>>> a
[82, 43, 65, 88, 43, 78]

>>> del a
>>> a
```

리스트 관련 함수

- count(x) - 리스트에 포함된 x요소의 개수 구하기

코드

```
>>> a=[1,2,3,4,3,2,3,2,2,2,3]

>>> a.count(1)
1

>>> a.count(2)
5

>>> a.count(3)
4
```

리스트 관련 함수

- all(리스트명) , any(리스트명)
  - all – 모든 요소가 참인 경우에만 True 리턴
  - any – 요소가 하나라도 참인 경우에 True 리턴

코드

```
>>> listdata1 = [0, 1, 2, 3, 4]
>>> listdata2 = [True, True, True]
>>> listdata3 = ['', [], (), {}, None, False]

>>> all(listdata1)
False
>>> all(listdata2)
True
>>> all(listdata3)
False
```

```
>>> any(listdata1)
True
>>> any(listdata2)
True
>>> any(listdata3)
False
```

숫자 0
빈 문자열 ''
빈 리스트 [ ]
빈 튜플 ( )
빈 딕셔너리 { }
None

이 값들은 모두 False

리스트 관련 함수

- 연결문자.join(리스트명) – 문자열 결합하기
  - 문자열이 요소인 리스트 인자를 받아 리스트의 모든 요소를 특정 문자열로 연결

코드

```
>>> join_list = ['2002년' , '06월', '29일']
>>> join_result = ""
>>> join_str = " "
>>> join_result = join_str.join(join_list)
>>> join_result
'2002년 06월 29일'

>>>s = "/"
>>> print(s.join(join_list))
'2002년/06월/29일'
```

리스트 내의 데이터가 문자열인 경우만 가능

리스트 관련 함수 정리

유형	사 용 방 법
메소드	리스트명.append(값), 리스트명.insert(인덱스, 값) 리스트명. index(값), 리스트명.count(값) 리스트명.sort() , 리스트명.reverse() 리스트명.clear() , 리스트명.remove(값) 리스트명.pop(), 리스트명.pop(인덱스) 연결문자.join(리스트명)
명령문	del 리스트명[인덱스] 또는 del 리스트명
내장함수	all(리스트명), any(리스트명) sum(리스트명), len(리스트명), max(리스트명), min(리스트명)

딕셔너리(**Dictionaries**) 조작하기

딕셔너리 이해하기

- { } 로 생성 var = { 'k1':10, 'k2': 20}
- dict({ })로 생성 var = dict({ 'k1':10, 'k2': 20})
- 'key : value'의 쌍으로 구성
  - 'key' - 중복 안 됨. 키는 정수, 문자열, 기타 파이썬 객체를 사용.
  - 'value' – 숫자, 문자열, 리스트, 딕셔너리, 튜플…
- key에 접근할 때는 '[ ]' 를 이용 var[k1] = 10
- 순서 없음
  - 인덱싱, 슬라이싱 안 됨
  - 정렬되지 않기 때문에 값을 추가하거나 검색할 때 빠른 응답을 제공

딕셔너리

- 기본 딕셔너리

코드

```
dic = {'name':'kim', 'phone':'0109993323', 'birth': '0115'}
```

key	value
name	kim
phone	01099993323
birth	0115

딕셔너리

- 새로운 요소 추가하기

코드

```
>>> a = {1: 'a'}

>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}

>>> a['name']='kim eun kyoung'
>>> a
{1: 'a', 2: 'b', 'name': 'kim eun kyoung'}
```

딕셔너리

- 특정 요소 값 수정하기

코드

```
>>> a
{1: 'a', 2: 'b', 'name': 'kim eun kyoung'}

>>> a['name']='jslee'

>>> a
{1: 'a', 2: 'b', 'name': 'jslee'}
```

딕셔너리

- del 딕셔너리명[key] : 특정 요소 값 삭제하기

코드

```
>>> a
{1: 'a', 2: 'b', 'name': 'jslee'}

>>> del a[1]
>>> a
{2: 'b', 'name': 'jslee'}

>>> del a
>>> a
Traceback (most recent call last):
  File "<pyshell#132>", line 1, in <module>
    a
NameError: name 'a' is not defined
```

딕셔너리

- clear() - 모든 요소 제거하기

코드

```
>>> a={1: 'a', 2: 'b', 'name': 'jslee'}

>>> a
{1: 'a', 2: 'b', 'name': 'jslee'}

>>> a.clear()
>>> a
{}
```

딕셔너리

- 리스트는 인덱스를 이용하여 요소 값을 얻어냈지만, 딕셔너리는 key를 이용해 Value를 얻어낸다.

코드

```
>>> fifa={ " 브라질":"1위", "대한민국":"28위", "스웨덴":"20위", "멕시코":"12위"}

>>> fifa["대한민국"]
'28위'

>>> fifa[ " 브라질"]
'1위'
```

딕셔너리

- 딕셔너리 만들 때 주의 사항1.
  - Key는 고유한 값이므로 중복되는 Key값을 설정해 놓으면 하나를 제외한 나머지는 모두 무시됨.

코드

```
>>> fifa={"브라질":"1위", "대한민국":"29위",   "대한민국":"28위"}
>>> fifa
{'대한민국': '28위', ' 브라질': '1위'}

>>> fifa={"브라질":"1위",   "대한민국":"28위", "대한민국":"29위"}
>>> fifa
{'대한민국': '29위', '브라질': '1위'}
```

딕셔너리

- 딕셔너리 만들 때 주의 사항2.
  - Key에 리스트는 쓸 수 없다(튜플은 Key로 쓸 수 있다)
  - Value에는 리스트를 쓸 수 있다.

코드

```
>>> a={"hi" : ["안녕", "니하오"]}
>>> a
{'hi': ['안녕', '니하오']}

>>> a={["안녕", "니하오"]:"hi"}
Traceback (most recent call last):
  File "<pyshell#158>", line 1, in <module>
    a={["안녕", "니하오"]:"hi"}
TypeError: unhashable type: 'list'
```

딕셔너리 관련 함수

- keys() - 키만 추출하기
  - Key만을 모아서 dict_keys라는 객체를 리턴

코드

```
>>> dic = {'name':'kim', 'phone':'0109993323', 'birth': '0115'}

>>> dic.keys()        #모든 요소를 추출하여 딕셔너리 뷰 객체로 리턴
dict_keys(['birth', 'phone', 'name'])
```

딕셔너리

- keys() - 키만 추출하기

코드

```
>>> fifa={"브라질":"1위", "대한민국":"28위", "스웨덴":"20위",
"멕시코":"12위"}

>>> for k in fifa.keys() :
        print(k)

브라질
대한민국
스웨덴
멕시코
```

딕셔너리

- values() - 값만 추출하기

코드

```
>>> fifa={"브라질":"1위", "대한민국":"28위", "스웨덴":"20위",
"멕시코":"12위"}

>>> for k in fifa.values() :
        print(k)


1위
28위
20위
12위
```

딕셔너리

- dic.keys(), dic.values()
  - 리스트처럼 만들어 반환을 하지만 정확하게 리스트는 아니다.
  - 인덱싱이나 슬라이싱은 불가능
  - 리스트 고유의 함수인 append, insert, pop, remove, sort등의 함수를 수행할 수는 없다.
  - 단, list()를 이용하여 리스트 타입으로 변경하면 인덱싱이나 슬라이싱 가능.

```
print(items[0])      #오류발생 : 인덱싱 안되므로


#다음과 같이 변경
dic_to_list =list(items)          #리스트로 변경하기
print(dic_to_list[0])             #인덱싱 출력
```

딕셔너리

- items() – key, values 쌍으로 모든 요소 추출하기
  - key와 value의 쌍을 튜플로 묶은 값을 dict_items 객체로 리턴.

코드

```
>>> fifa
{"브라질":"1위", "대한민국":"28위", "스웨덴":"20위", "멕시코":"12위"}

>>> fifa_list = fifa.items()
>>> print(fifa_list)
dict_items([('브라질', '1위'), ('대한민국', '28위'), ('스웨덴', '20위'), ('멕시코', '12위')])

>>> for i in fifa_list :
        print(i)

('브라질', '1위')
('대한민국', '28위')
('스웨덴', '20위')
('멕시코', '12위')
```

딕셔너리

- 특정 키가 존재하는지 확인하기 - '[키 in 딕셔너리명]' 으로
  - 존재하면 True, 없으면 False 반환
  - 문자열과 리스트에서도 사용 가능

코드

```
name_dic={'민준':13869,'지훈':13376, '현우':10739, '은경':10335}
k=input('이름을 입력하세요 : ')

if k in name_dic:
    print('이름이 <%s>인 출생아 수는 <%d>명입니다.' %(k, name_dic[k]))
else :
    print('자료에는 <%s>인 이름은 존재하지 않습니다.' %k)
```

실행결과

```
이름을 입력하세요 : 현우
이름이 <현우>인 출생아 수는 <10739>명입니다.
```

딕셔너리

- get()
  - key로 Value를 얻고자 할 때 사용.
  - 존재하지 않는 키의 값에 접근할 때 에러를 발생시키지 않고 None을 리턴

코드

```
name_dic={'민준':13869,'지훈':13376, '현우':10739, '은경':10335}
print(name_dic['은경'])              #존재하는 키의 값을 출력
print(name_dic['점숙'])              #오류발생, 존재하지 않은 키의 값을 출력
```

< 수정 >

```
print(name_dic.get('점숙'))    또는
print(name_dic.get('점숙', 'default value'))
```

실행결과

```
10335
None
default value
```

튜플(**Tuples**) 조작하기

튜플 이해하기

- ( ) 로 생성 var = (1, 2, 3)
- tuple(( ))로 생성 var=tuple((1, 2, 3))
- 튜플의 값이 1개만 있을 경우는 콤마(,)를 붙여야 함
  var = (1,)
- 수정, 삭제 할 수 없음
  - 프로그램이 실행되는 동안 그 값이 항상 변하지 않기를 바란다거나 값이 바뀔까 걱정하고 싶지 않다면 튜플을 사용해야 한다.
- 값에 접근할 때는 '[ ]' 를 이용
- 인덱싱과 슬라이싱은 리스트와 동일
- 연결과 반복 연산도 리스트와 동일

튜플

- 기본 튜플

코드

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

튜플

- 인덱싱과 슬라이싱

코드

```
>>> t1 = (3, 5, 'korea', 'Big')
>>> t1[0]
3
>>> t1[3]
'Big‘

>>> t1[1:3]
(5, 'korea')
```

튜플

- 연결과 반복

코드

```
>>> t1=('대한민국 ')
>>> t2=("Fighting ")
>>> t3=('!!!~~~~')
>>> t1 + t2 + t3
'대한민국 Fighting !!!~~~~'

>>> t1 + t2 * 3
'대한민국 Fighting Fighting Fighting'
```