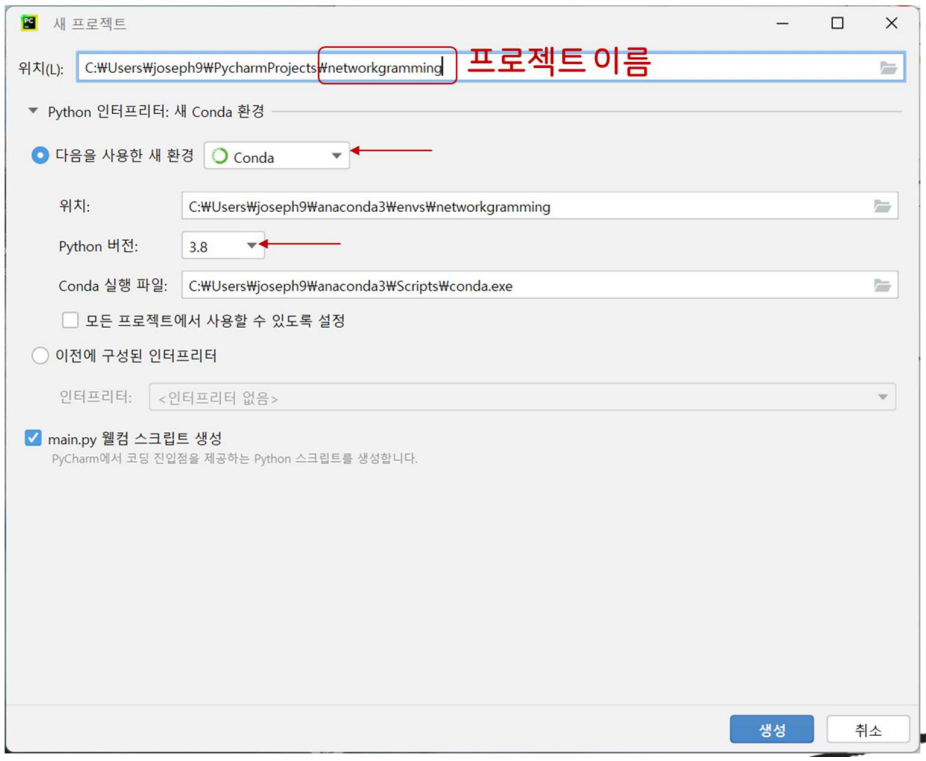
01. 파이썬 시퀀스, 매핑 타입 및 예외처리

실습 환경

실습 환경 SW

- PyCharm(www.jetbrains.com)
- Anaconda(www.anaconda.com)
- python 버전 : 3.8(최신 ~~3.10~~)

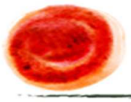
PyCharm 새 프로젝트 생성



파이썬의 SEQEUNCE

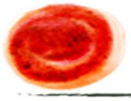
시퀀스, SEQUENCE

- 데이터를 나열한 것
  - 'good', b'₩xe2₩x99₩xa5', [1, 2, 3], (1, 2, 3) 등
- 각 데이터는 나열 순서에 따른 인덱스로 접근
- 종류
  - 변경 가능한 시퀀스,mutable sequence
  - 변경 불가능한 시퀀스, immutable sequence



변경 가능한 시퀀스, mutable sequence

- 시퀀스 내 데이터를 변경, 삽입, 삭제 가능
- 리스트, list()



변경 불가능한 시퀀스, immutable sequence

- 시퀀스 내 데이터 변경이 불가능
- 종류
  - 문자열(string)
  - 튜플, tuple()
  - 바이트, bytes()

리스트

- 데이터를 순차 나열하여 저장
- 종류 제한없는 데이터.

$$vaiable = [\ value1,\ value2,\ \cdots]$$

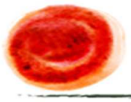
- 인덱스를 이용하여 데이터에 접근

$$vaiable[index]$$

- 반복문을 이용한 데이터 처리 가능

리스트 사용 예

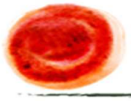
```
1  data = [1, 'hello', 3.14, True, ['hello', 'hi', 'nice']]
2  print(data)
3  data[2] = 2.718
4  data[0] = data[0] + 1
5  print(data[0], data[1], data[2], data[3])
6  print(data[4])
7  print(data[4][1])
8  data2 = []
9  data3 = list()
10 print(data2, data3)
11 print(len(data), len(data2))
```



반복문을 이용한 리스트 사용(1/3)

- 인덱스를 이용하는 방법

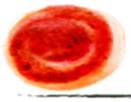
```
1 lst = [2, 4, 6, 8, 10]
2 total = 0
3 for idx in range(len(lst)):
4     total = total + lst[idx]
5 print('Total in list : ', total)
```



반복문을 이용한 리스트 사용(2/3)

- 리스트의 원소를 순서대로 이용하는 방법

```
for item in lst:
    block          # 반복 실행할 블록
else :
    else_block     # 반복 실행 완료 후 실행되는 블록
```



반복문을 이용한 리스트 사용(3/3)

- 리스트의 원소를 순서대로 이용하는 방법

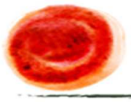
```
1  data = list(range(100))
2  o_total, o_count = 0, 0
3  e_total, e_count = 0, 0
4  for datum in data:
5      if datum %2 == 0:
6          e_total = e_total + datum
7          e_count = e_count + 1
8      else:
9          o_total = o_total + datum
10         o_count = o_count + 1
11 print('Even Total & Average : ', e_total, e_total / e_count)
12 print('Odd Total & Average : ', o_total, o_total / o_count)
```

슬라이스, slice(1/2)

- 인덱스 범위를 이용한 부분 리스트

vaiable[*begin_idx* : *end_idx* : *increment_idx*]

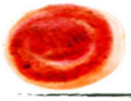
```
1  data = ['good', 'morning', 'hello', 'welcome']
2  print(data[1:4:1])
3  print(data, data[::])
4  print(data[1:4], data[1:4][0])
5  print(data[:4])
6  print(data[1:])
7  print(data[::2])
8  print(data[::-1])
9  data[1:3] = ['bye']
10 print(data)
```



슬라이스, slice(2/2)

- 역순 리스트 범위

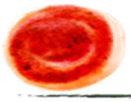
```
1 numbers = list(range(1,11))
2 numbers[3:0:-1] = [11, 12, 13]
3 print(numbers)
4 for n in numbers[::-1]:
5     print(n, end='␣')
6 print()
7 print(numbers[2:2:-1])
8 print(numbers[2:1:-1])
```



리스트 관련 메소드

- 리스트가 제공한 기능 이용
  - 리스트 객체를 이용한 메소드 호출

```
lst.method(parameter)
```



리스트 메소드

메소드	내용	사용예
lst.**append**(ele)	새 데이터 *ele*를 리스트에 추가	lst.append('good')
lst.**extend**(list)	리스트 *list*를 리스트 끝에 붙여 확장	lst.extend([1, 2, 3])
lst.**insert**(idx, ele)	인덱스 idx 위치에 데이터 ele 삽입	lst.insert(1, 'morning')
lst.**remove**(ele)	리스트에서 첫번째 ele 삭제	lst.remove(1)
lst.**clear**()	리스트의 모든 데이터 제거	lst.clear()
lst.**pop**(*[idx]*)	마지막 또는 인덱스 idx의 데이터를 리스트에서 제거하고, 반환	lst.pop()
lst.**count**(ele)	리스트에 데이터 *ele*의 갯수	lst.count(1)
lst.**reverse**()	데이터의 저장 순서를 역순으로 변경	lst.reverse()
lst.**copy**()	리스트이 복사본을 반환	lst.copy()
lst.**index**(ele*[,[s, e]]*)	리스트 내 데이터 ele의 인덱스를 반환. *s*와 *e*는 인덱스 범위와 동일하게 동작하며 생략 가능.	lst.index('good',2)
lst.**sort**()	리스트 내 데이터를 크기 순으로 정렬	lst.sort()
lst.**sort**(key=None)	정렬 기준 값을 반환하는 함수 지정	lst.sort(key=getkey)
lst.**sort**(reverse=False)	데이터의 크기 역순으로 정렬	lst.sort(reverse=True)

리스트 메소드 호출의 예:

```
1  def keyvalue(d):
2      return d[1]
3  lst = ['s1', 'r2', 'a3', 'c4']
4  lst.append('m5')
5  lst.extend(['e6', 'd7'])
6  print('lst 1 : ', lst)
7  lst2 = lst
8  lst3 = lst.copy()
9  lst.reverse()
10 print('lst 2 : ', lst2, 'lst 3 : ', lst3)
11 lst.sort()
12 lst3.sort(key=keyvalue)
13 print('lst 2 : ', lst2, 'lst 3 : ', lst3)
```

튜플

- 0 또는 그 이상의 데이터를 순차 나열하여 저장
- 인덱스를 이용한 데이터 접근
- 내용 변경이 불가능

vaiable = (*value1*, *value2*, ···)

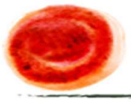
vaiable = **tuple()** or **tuple**(*list*) or **tuple**(*tuple*)

- 튜플 메소드

메소드	내용	사용예
lst.**count**(ele)	튜플 내 데이터 *ele*의 갯수	tpl.count(1)
lst.**index**(ele*[,[s, e]]*)	튜플 내 데이터 ele의 인덱스를 반환. *s*와 *e*는 인덱스 범위와 동일하게 동작하며 생략 가능.	tpl.index('good',2)

튜플 사용의 예

```
1  def getTotal(lst):
2      lst = tuple(lst)
3      total = 0
4      for v in lst:
5          total = total + v
6      return total
7  print(getTotal(()))
8  print(getTotal([1, 2, 3, 4, 5]))
9  print(getTotal((2, 4, 6, 8)))
10 print(getTotal(range(1,5)))
11 data=(1,)
12 print(getTotal(data))
13 tpl = ('good', 'morning', 'hello')
14 print(tpl.count('hello'))
15 print(('good', 'morning', 'hello').index('hello'))
```



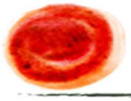
바이트열

- 바이트 값의 나열
  - ASCII 문자 또는 x00 ~ xFF 사이의 값

$$variable = \mathbf{b}\text{'byte}\cdots\text{'}$$

- 인덱스를 이용한 바이트 접근

$$variable[index]$$



바이트열 사용 예:

```
1  az = b'AtoZ'
2  heart = bytes(b'\xe2\x99\xa5')
3  print(az)
4  print(heart)
5  print(heart.decode())
6  print(az[1])
7  print(heart[1], hex(heart[1]))
```

파이썬의 MAPPING

매핑, mapping

- 인덱스 집합에 대응하는 값의 집합으로 구성
  - 키 : 인덱스 집합의 원소
  - 값 : 키에 대응하는 객체
- 종류
  - 딕셔너리(dictionary), dict()

딕셔너리, dict()

- 키(key)에 값(value)를 사상(mapping)시켜 저장
- '순서'의 개념 X

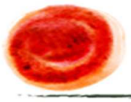
variable = { *key1:value1*, *key2:value2*, ···}
variable = **dict**() or **dict**(*[(key1,value1), (key2, value2), ···]*)
variable = **dict**(*key1=value1*, ···)

- 키를 이용한 데이터 접근

variable[*key*]
variable[*key*] = new_value

딕셔너리 생성의 예

```
1  ata1 = {'kor':95, 'eng':90, 'name':'hong'}
2  data2 = dict([('kor', 95), ('eng', 90), ('name', 'hong')])
3  data3 = dict(kor=95, eng=90, name='hong')
4  print(data1)
5  print(data1['kor'])
6  data1['kor'] = 100
7  print(data1)
```

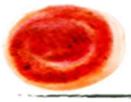


딕셔너리 메소드

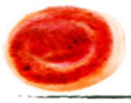
메소드	내용	사용예
dic.**copy**()	딕셔너리의 복사본을 반환	dic.copy()
dic.**keys**()	딕셔너리가 저장하고 있는 모든 키를 반환	dic.keys()
dic.**values**()	딕셔너리가 저장하고 있는 모든 값을 반환	dic.values()
dic.**items**()	딕셔너리가 저장하고 있는 모든 (key, value) 쌍을 반환	dic.items()
dic.**pop**(*key*)	키의 값을 딕셔너리에서 제거하고 반환	dic.pop('kor')
dic.**clear**()	디셔너리의 모든 데이터 제거	dic.clear()

반복문 이용한 딕셔너리 이용

```
1 data1 = {'kor':95, 'eng':90, 'name':'hong'}
2 for key in data1.keys():
3     print(data1[key])
4 for key, value in data1.items():
5     print(key, '—>', value)
```



파이썬의 EXCEPTION HANDLING



실행 시간 오류, runtime error

- 프로그램이 정상 실행되지 못하도록 하는 오류
  - 인덱스 범위를 벗어난 인덱스 사용
  - 존재하지 않는 파일에서 데이터 읽기 시도, …

예외, exception

- 실행 시간 오류로 인하여 프로그램 실행이 중단되는 경우에 발생
  - 다양한 예외 발생 원인
- 프로그램 성공 실행은 예외 발생 X
- 실행 시간 오류를 위한 간단한 예

```
1  n = int(input('input number : '))
2  print(100/n)
```

정상 실행 및 예외 발생의 예

```
input number : 5
20.0
```

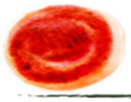
(a) 정상 실행

```
input number : 0
Traceback (most recent call last):
  File "C:\Utst.py", line 2, in <module>
    print(100/n)
ZeroDivisionError: division by zero
```

(b) 0으로 나눈 오류

```
input number : 4a
Traceback (most recent call last):
  File "C:\Utst.py", line 1, in <module>
    n = int(input('input number : '))
ValueError: invalid literal for int() with base 10: '4a'
```

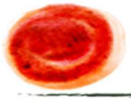
(c) 잘못된 입력으로 인한 오류



예외 처리, exception handing

- 기본 – 오류 메시지 출력 후 프로그램 종류
- 예외를 발생시킨 프로그램에서 처리

```
try :
    try-block
except [exception-type [as identifier ]]:
    handler-block
finally :
    finally-block
```



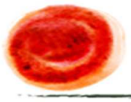
예외 처리의 예

```
1  try:
2      n = int(input('input number : '))
3      print(100/n)
4  except ZeroDivisionError as ev:
5      print('Zero is not be divisor.', ev)
6  except ValueError:
7      print('Your input is not converted to integer.')
```

실습 내용

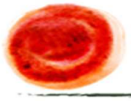
실습 #1

- random 모듈을 이용하여 0~100사이의 정수 난수 100 개를 생성하여 리스트에 저장하라.
- 리스트에 저장된 정수 중 가장 큰 수와 작은 수를 찾아 출력하라.
- 리스트의 메소드를 이용하여 가장 큰 수와 작은 수의 인덱스를 출력하라.



실습 #2

- 사용자로 부터 이름, 나이, 키, 성적을 입력 받아 각각 name, age, height, score를 키로 하여 딕셔너리에 저장하라.
- 딕셔너리에 저장된 모든 키에 대해 각각의 값을 출력하라.



실습 #3

- input()을 이용하여 사용자 입력을 받은 후 정수 값으로 변환할 때 숫자가 아닌 문자가 입력되면 정수로 변환하는 과정에 ValueError 예외가 발생한다. 반복문과 예외 처리를 이용하여 정상적으로 정수 변환이 가능하도록 프로그래밍 하라.

Q&A