

# Python 과학 프로그래밍 기초

## 시퀀스형 데이터 (1)

한림대학교 소프트웨어융합대학  
박섭형

2021년 1학기

### 1 시퀀스 형 데이터

- Range
- List
- Tuple
- String
- bytes
- bytearray

#### 1.1 리스트(List)와 튜플 형

리스트와 튜플의 특징

- 리스트
  - Sequence 형
  - Mutable (추가 및 수정 가능)
  - 정렬 가능
  - 범용
  - Python 데이터 유형 중에서 가장 널리 사용됨
  - 리스트 내의 원소의 개수는 제한이 없고, 원소는 어떤 유형의 데이터도 사용 가능
- 튜플
  - Sequence 형
  - 리스트보다 처리 속도가 빠름
  - Immutable (추가 및 수정 불가능)

- 정렬 불가능
- 고정형 데이터에 적합
- 튜플 내의 원소의 개수는 제한이 없고, 원소는 어떤 유형의 데이터도 사용 가능

## 리스트 구문

- [ ] 안에 항목(또는 원소)를 쉼표(,)로 분리

```
a = [1, 3, 5]
```

## 리스트 생성 방법

### 빈 리스트

```
[1]: list1 = []  
list1
```

```
[1]: []
```

```
[2]: list2 = list()  
list2
```

```
[2]: []
```

### 단일 원소 리스트

```
[3]: list_s = [1]  
list_s
```

```
[3]: [1]
```

### 정수 리스트

```
[4]: list_i = [1, 3, 5]  
list_i
```

```
[4]: [1, 3, 5]
```

## 실수 리스트

```
[5]: list_f = [1., 3., 5.]  
list_f
```

```
[5]: [1.0, 3.0, 5.0]
```

## 복합 리스트

```
[6]: list_cmplx = [1., '3', [1, 3, 'text']]  
list_cmplx
```

```
[6]: [1.0, '3', [1, 3, 'text']]
```

## list() 함수 이용

```
[7]: list3 = list(range(5))  
list3
```

```
[7]: [0, 1, 2, 3, 4]
```

```
[8]: list3 = list("abc")  
list3
```

```
[8]: ['a', 'b', 'c']
```

## 리스트 컴프리헨션(Comprehension, 조건제시법)

```
[9]: list4 = []  
for n in range(5):  
    list4.append(n)  
list4
```

```
[9]: [0, 1, 2, 3, 4]
```

```
[10]: list5 = [n for n in range(5)]  
list5
```

```
[10]: [0, 1, 2, 3, 4]
```

```
[11]: x = [n for n in range(5)]  
y = [n**2 for n in range(5)]
```

```
print(x)
print(y)
```

```
[0, 1, 2, 3, 4]
[0, 1, 4, 9, 16]
```

```
[12]: x = [n for n in range(10) if n % 3]
x
```

```
[12]: [1, 2, 4, 5, 7, 8]
```

### Nested List

```
[13]: list_n = [[1,2,3], [4,5,6]]
list_n
```

```
[13]: [[1, 2, 3], [4, 5, 6]]
```

### 튜플 구문

- () 안에 항목(또는 원소)를 쉼표(,)로 분리

```
b = (1, 3, 5)
```

### 튜플 생성 방법

#### 빈 튜플

```
[14]: tuple1 = ()
tuple1
```

```
[14]: ()
```

```
[15]: tuple2 = tuple()
tuple2
```

```
[15]: ()
```

## 단일 원소 튜플

```
[16]: tuple_s = (1,)  
tuple_s
```

[16]: (1,)

[ ]:

## 정수 튜플

```
[17]: tuple_i = (1, 3, 5)  
tuple_i
```

[17]: (1, 3, 5)

## 실수 튜플

```
[18]: tuple_f = (1., 3., 5.)  
tuple_f
```

[18]: (1.0, 3.0, 5.0)

## 복합 튜플

```
[19]: tuple_cmplx = (1., '3', (1, 3, 'text'))  
tuple_cmplx
```

[19]: (1.0, '3', (1, 3, 'text'))

## tuple() 함수 이용

```
[20]: tuple3 = tuple(range(5))  
tuple3
```

[20]: (0, 1, 2, 3, 4)

```
[21]: tuple4 = tuple("abc")  
tuple4
```

[21]: ('a', 'b', 'c')

```
[22]: tuple5 = tuple([1, 3, 5])
tuple5
```

```
[22]: (1, 3, 5)
```

튜플 컴프리헨션은 없음

```
[23]: x = (n for n in range(5))
print(x)
```

```
<generator object <genexpr> at 0x000002094B40C5F0>
```

```
[24]: next(x)
```

```
[24]: 0
```

```
[25]: x = tuple([n for n in range(5)])
print(x)
```

```
(0, 1, 2, 3, 4)
```

## 1.2 시퀀스 자료형 다루기

시퀀스 자료형의 원소 접근

정수 인덱스를 통한 시퀀스 자료형의 원소 접근

- index를 통해서 list의 원소에 접근
- index는 정수이며 첫 원소의 index는 0
- 뒤에서부터 index를 부여할 때는 -1, -2, ...

lst = ['a', 'b', 'c', 'd', 'e']일 경우

list	'a'	'b'	'c'	'd'	'e'
index	0	1	2	3	4
index	-5	-4	-3	-2	-1

리스트

```
[26]: l = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
print(l[0])
print(l[3])
print(l[-2])
```

a  
d  
f

### 튜플

```
[27]: t = (0, 2, 4, 6, 8, 10, 12, 14)
print(t[0])
print(t[3])
print(t[-2])
```

0  
6  
12

### 스트링

```
[28]: s = "abcdefg"
print(s[0])
print(s[3])
print(s[-2])
```

a  
d  
f

### Range

```
[29]: r = range(7)
print(r[0])
print(r[3])
print(r[-2])
```

```
0  
3  
5
```

### 인덱스 슬라이싱 통한 시퀀스 자료형의 원소 접근

```
[30]: print(l[1:3])  
      print(t[1:3])  
      print(s[1:3])  
      print(r[1:3])
```

```
['b', 'c']  
(2, 4)  
bc  
range(1, 3)
```

```
[31]: print(l[::-2])  
      print(t[::-1])  
      print(s[::-2])  
      print(r[::-2])
```

```
['a', 'c', 'e', 'g']  
(14, 12, 10, 8, 6, 4, 2, 0)  
geca  
range(0, 7, 2)
```

### 슬라이스 객체를 이용한 시퀀스 자료형의 원소 접근

```
slice(stop)  
slice(start, stop[, step])
```

```
[32]: idx = slice(2,8,2)  
      print(l[slice(4)])  
      print(t[idx])  
      print(s[idx])  
      print(r[idx])
```

```
['a', 'b', 'c', 'd']
(4, 8, 12)
ceg
range(2, 7, 2)
```

### 중첩 리스트/튜플의 인덱싱

```
[33]: a = [[1, 2], [3, 4, 5]]
a[1][1]
```

```
[33]: 4
```

```
[34]: a = [[1,3,2], (3,4,5), 'abc']
a[2][1]
```

```
[34]: 'b'
```

### 리스트 원소 변경 및 제거

```
[35]: a = list(range(1,5)) # a = [1, 2, 3, 4]
a[0:2] = [11, 12]      # 일부 원소 변경
a
```

```
[35]: [11, 12, 3, 4]
```

```
[36]: a[0:2] = []          # 일부 원소 제거
a
```

```
[36]: [3, 4]
```

```
[37]: a[:0] = a           # 제일 앞에 자기 삽입
a
```

```
[37]: [3, 4, 3, 4]
```

```
[38]: a[1:1] = [31, 32]    # 일부 원소 삽입
a
```

```
[38]: [3, 31, 32, 4, 3, 4]
```

```
[39]: a[1] = [31, 32]      # 단일 원소 변경
a
```

```
[39]: [3, [31, 32], 32, 4, 3, 4]
```

```
[40]: a = list(range(1,8))
a
```

```
[40]: [1, 2, 3, 4, 5, 6, 7]
```

```
[41]: del a[2]
a
```

```
[41]: [1, 2, 4, 5, 6, 7]
```

```
[42]: a[:] = []
a
```

```
[42]: []
```

```
[43]: a = [1, 3, 5]
a = []
a
```

```
[43]: []
```

## 시퀀스 자료형의 연산

+, \* 연산자를 이용한 이어붙이기 (concatenation)

```
[44]: a = [1, 2]
b = [10, 11]
a + b
```

```
[44]: [1, 2, 10, 11]
```

```
[45]: c = (1, 2)
d = 3 * c
d
```

```
[45]: (1, 2, 1, 2, 1, 2)
```

```
[46]: s = "abc"
b = s + s
b
```

```
[46]: 'abcabc'
```

```
[47]: 5 * s
```

[47]: 'abcabcabcaabc'

### 시퀀스 자료형의 멤버십 확인

```
[48]: l = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
      t = (0, 2, 4, 6, 8, 10, 12, 14)
      s = "abcdefg"
      r = range(7)
```

[49]: 'a' in l

[49]: True

[50]: 'b' not in t

[50]: True

[51]: 'cd' in s

[51]: True

[52]: 8 not in r

[52]: True

### 시퀀스 자료형 메쏘드

#### 시퀀스 자료형 공통 메쏘드

##### 시퀀스 내 원소의 인덱스 구하기

```
[53]: l = [1, 5, 8, 3, 2, 4, 8, 3, 5]
      print(l.index(8))
      print(l.index(8, 3))
      print(l.index(8, 3, 6))
```

2

6

ValueError

Traceback (most recent call last)

```
<ipython-input-53-8f12803c9fd6> in <module>
      2 print(l.index(8))
      3 print(l.index(8, 3))
----> 4 print(l.index(8, 3, 6))

ValueError: 8 is not in list
```

### 시퀀스 내 원소의 갯수 구하기

```
[ ]: l = [1, 5, 8, 3, 2, 4, 8, 3, 5]
      print(l.count(8))
```

```
[ ]: s = "Why not the best?"
      print(s.count('t'))
```

### 리스트 메쏘드

#### Append:

`list.append(object)`: 리스트의 뒤에 object를 추가

```
[54]: a = [3, 5, 7]
      a.append(8)
      a
```

[54]: [3, 5, 7, 8]

```
[55]: a.append([8])
      a
```

[55]: [3, 5, 7, 8, [8]]

```
[56]: list.append(a, 9)
      a
```

[56]: [3, 5, 7, 8, [8], 9]

```
[57]: list.append(a, (10, 11))
      a
```

[57]: [3, 5, 7, 8, [8], 9, (10, 11)]

Extend:

`list.extend(iterable)`: 리스트의 뒤에 iterable의 원소들을 추가

```
[58]: a = [3, 5, 7]
      a.extend([8])
      a
```

[58]: [3, 5, 7, 8]

```
[59]: list.extend(a, "abc")
      a
```

[59]: [3, 5, 7, 8, 'a', 'b', 'c']

```
[60]: a.extend((1,3))
      a
```

[60]: [3, 5, 7, 8, 'a', 'b', 'c', 1, 3]

```
[61]: list.extend(a, [1,2])
      a
```

[61]: [3, 5, 7, 8, 'a', 'b', 'c', 1, 3, 1, 2]

```
[62]: a.extend({'a':3, 'b':5})
      a
```

[62]: [3, 5, 7, 8, 'a', 'b', 'c', 1, 3, 1, 2, 'a', 'b']

Insert:

`list.insert(index, object)`: 리스트의 index 바로 앞에 object를 삽입

```
[63]: b = [1, 3, 5]
      b.insert(2, 10)
      b
```

[63]: [1, 3, 10, 5]

```
[64]: b.insert(-1, 6)
      b
```

[64]: [1, 3, 10, 6, 5]

```
[65]: list.insert(b, 2, 9)
b
```

[65]: [1, 3, 9, 10, 6, 5]

Pop:

list.pop(index=-1): index 위치의 item을 제거하고 그 item을 반환

```
[66]: a = [1, 3, 1, 3, 5, 2]
b = a.pop()
print(a, b)
```

[1, 3, 1, 3, 5] 2

```
[67]: b = list.pop(a)
print(a, b)
```

[1, 3, 1, 3] 5

```
[68]: b = a.pop(2)
print(a, b)
```

[1, 3, 3] 1

Remove:

list.remove(value): 첫번째 value를 제거. 반환값 없음

```
[69]: a = [1, 3, 1, 3, 5, 2]
a.remove(1)
a
```

[69]: [3, 1, 3, 5, 2]

```
[70]: list.remove(a, 1)
a
```

[70]: [3, 3, 5, 2]

Clear:

`list.clear()`: 리스트의 모든 항목들을 제거하고 빈 리스트로 만듦

```
[71]: a = [1, 3, 1, 3, 5, 2]
a.clear()
a
```

[71]: []

Sort:

`list.sort()`: 리스트의 원소들의 순서를 오름 차순으로 재배치

```
[72]: a = [1, 3, 1, 3, 5, 2]
a.sort()
a
```

[72]: [1, 1, 2, 3, 3, 5]

```
[73]: a = ['f', 'fix', 'abs', 'abc']
a.sort()
a
```

[73]: ['abc', 'abs', 'f', 'fix']

```
[74]: a = ['f', 'fix', 'abs', 'abc']
list.sort(a)
a
```

[74]: ['abc', 'abs', 'f', 'fix']

```
[75]: a = [1, 5, 3, -1]
a.sort()
a
```

[75]: [-1, 1, 3, 5]

Reverse:

`list.reverse()`: 리스트의 원소들의 순서를 반대로 재배치

```
[76]: a = [1, 3, 1, 3, 5, 2]
a.reverse()
a
```

[76]: [2, 5, 3, 1, 3, 1]

```
[77]: a = [1, 3, 1, 3, 5, 2]
list.reverse(a)
a
```

[77]: [2, 5, 3, 1, 3, 1]

```
[78]: a = ['f', 'fix', 'abs', 'abc']
a.reverse()
a
```

[78]: ['abc', 'abs', 'fix', 'f']