

PYTHON PROGRAMMING
LECTURE 3: DATA STRUCTURE

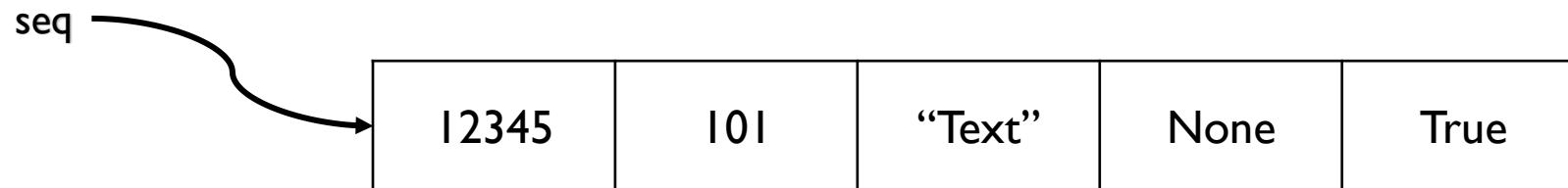
goorm

KAIST AI
Graduate School of AI

 DAVIAN

List

- 배열: 일련의 데이터를 하나로 묶음



- 파이썬 배열의 특징

- 대괄호로 선언 “[value1, value2, …]”
- 아무 타입이나 넣기 가능
- 길이가 정해져 있지 않음

```
>>> seq = [12345, 101, "Text", None, True]  
>>> seq  
[12345, 101, 'Text', None, True]
```

List Index & Slicing

```
>>> seq = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> seq[0]      # 첫번째 요소
1
>>> seq[8]
9
>>> seq[-1]     # 뒤에서 첫번째 요소
10
>>> seq[-3]
8
```

- **List indexing**

- `seq[index]` 형태로 요소 하나 접근
- 0부터 숫자세기 시작
- 음수 가능 (뒤에서부터 접근)

- **List Slicing**

- `seq[start: end: step]` 형태로 List 자르기
- `end`번째는 포함하지 않음

```
>>> seq[:3]          # 앞 3번째 전까지
[1, 2, 3]
>>> seq[3:]          # 앞 3번째 부터
[4, 5, 6, 7, 8, 9, 10]

>>> seq[3:-1]         # 앞 3번째 부터 뒤에 1번째 전까지
[4, 5, 6, 7, 8, 9]
>>> seq[-3:-1]
[8, 9]

>>> seq[::-2]          # 두 칸씩 뛰면서
[1, 3, 5, 7, 9]
>>> seq[9:2:-1]        # 한 칸씩 뒤로 가면서
[10, 9, 8, 7, 6, 5, 4]
>>> seq[-2:2:-1]
[9, 8, 7, 6, 5, 4]
```

List Operators

리스트간 사칙연산 가능.

```
>>> a = [1, 2, 3, 4]
>>> b = [5, 6, 7, 8]
>>> a + b                      # 리스트 합치기
[1, 2, 3, 4, 5, 6, 7, 8]

>>> a[0] = 'something?'          # 리스트 내에 값 바꾸기

>>> a * 2                        # 곱하기로 여러 개를 여러 번 합치기
['something?', 2, 3, 4, 'something?', 2, 3, 4]

>>> 'something?' in a           # 리스트 안에 요소가 있는지 확인
True
```

List Operators

```
>>> seq = [1, 2, None, True]
>>> len(seq)          # 리스트 길이 구하기, 내장 함수 형태, seq.length가 아니다!
4

>>> seq.append("something")      # 맨 뒤에 요소 추가, 메소드 형태
>>> seq.extend([5, 6])         # 맨 뒤에 리스트 추가, seq += [5, 6]
>>> seq.insert(1, 1.5)         # 원하는 곳에 삽입, 1번째에 1.5 삽입, (index, val)
>>> seq
[1, 1.5, 2, None, True, 'something', 5, 6]

>>> del seq[1]                # 1번째 요소 삭제, 예약어 형태, seq.delete가 아니다!
>>> seq.remove('something')    # 원하는 값을 삭제
>>> seq
[1, 2, None, True, 5, 6]
```

Reserved words & Built-in Functions & Methods

파이썬에서 제공되는 기능은 일반적으로 다음과 같은 형태

예약어

- 일종의 문법적인 요소
- 괄호를 쓰지 않음
- 재정의 불가능

- 예시

- del
- if ... else ...
- assert

내장함수

- 기본 정의된 함수
- 별개의 함수 사용
- 재정의 가능
- 편의성 향상

- 예시

- len()
- sum()
- range()

메소드

- 객체 내에 정의된 함수
- .method() 으로 접근
- Overriding
- 해당 객체를 다룸

- 예시

- .append()
- .insert()
- .extend()

Reserved words

Python Keywords

False	def	if	raise
-------	-----	----	-------

None	del	import	return
------	-----	--------	--------

True	elif	in	try
------	------	----	-----

and	else	is	while
-----	------	----	-------

as	except	lambda	with
----	--------	--------	------

assert	finally	nonlocal	yield
--------	---------	----------	-------

break	for	not
-------	-----	-----

class	from	or
-------	------	----

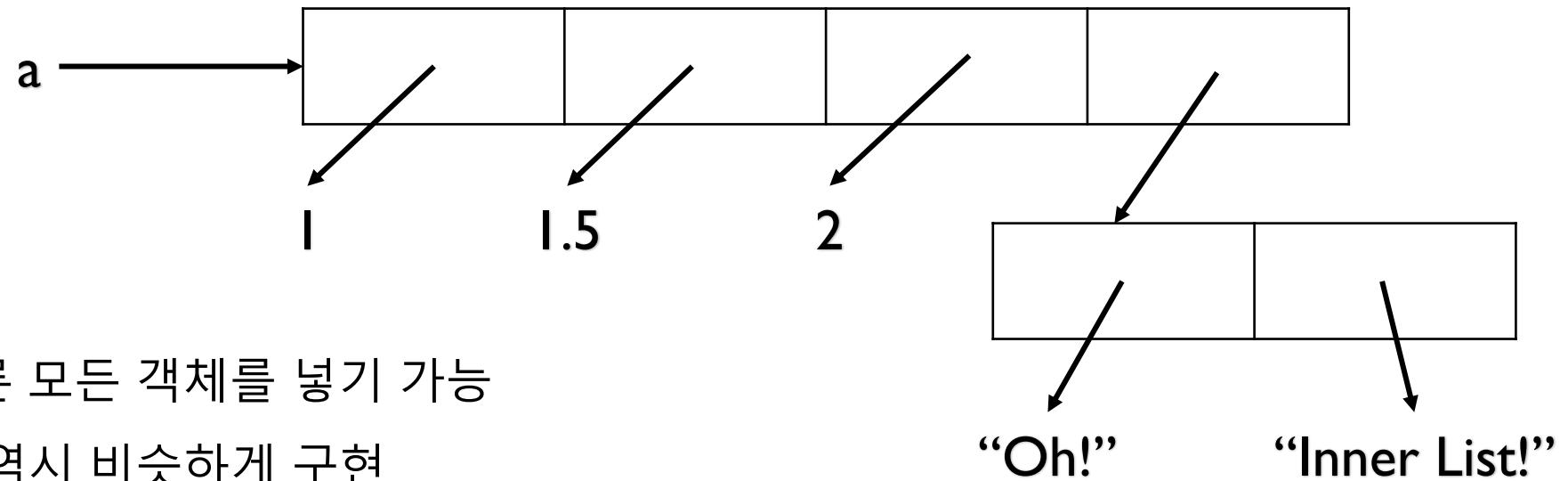
continue	global	pass
----------	--------	------

Built-in Functions

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Memory Structure of List

```
>>> a = [1, 1.5, 2, ["Oh!", "Inner List!"]]      # List in list  
>>> a  
[1, 1.5, 2, ['Oh!', 'Inner List!']]  
  
>>> a = [[1, 2, 3], [4, 5, 6]]                  # 이차원 리스트  
>>> a  
[[1, 2, 3], [4, 5, 6]]
```



List as Mutable Data Type

리스트 등은 가변 타입임으로 주의!

```
>>> a = [1, 2, 3, 4, 5]      # 가변 타입
>>> b = a                    # 같은 주소를 가르침
>>> a += [6]                 # In-place 연산, a = a + [6] 과 다름
>>> b
[1, 2, 3, 4, 5, 6]

>>> a = 'Something'          # 불변 타입
>>> b = a                    # 같은 주소를 가르침
>>> a += '?'
# 불변 타입이기에 Out-place, a = a + '?
>>> b
'Something'
>>> a
'Something?'
```

Time Complexity of List

Operation	Example	Big-O	Notes
index	<code>l[i]</code>	$O(1)$	
store	<code>l[i] = 0</code>	$O(1)$	
length	<code>len(l)</code>	$O(1)$	
append	<code>l.append(5)</code>	$O(1)$	
pop	<code>l.pop()</code>	$O(1)$	<code>l.pop(-1)</code> 과 동일
clear	<code>l.clear()</code>	$O(1)$	<code>l = []</code> 과 유사
slice	<code>l[a:b]</code>	$O(b-a)$	<code>l[:]</code> : $O(\text{len}(l)-0) = O(N)$
extend	<code>l.extend(...)</code>	$O(\text{len}(...))$	확장 길이에 따라
construction	<code>list(...)</code>	$O(\text{len}(...))$	요소 길이에 따라
check <code>==</code> , <code>!=</code>	<code>l1 == l2</code>	$O(N)$	비교
insert	<code>l.insert(i, v)</code>	$O(N)$	<code>i</code> 위치에 <code>v</code> 를 추가
delete	<code>del l[i]</code>	$O(N)$	
remove	<code>l.remove(...)</code>	$O(N)$	
containment	<code>x in/not in l</code>	$O(N)$	검색
copy	<code>l.copy()</code>	$O(N)$	<code>l[:]</code> 과 동일 - $O(N)$
pop	<code>l.pop(i)</code>	$O(N)$	<code>l.pop(0):O(N)</code>
extreme value	<code>min(l)/max(l)</code>	$O(N)$	검색
reverse	<code>l.reverse()</code>	$O(N)$	그대로 반대로
iteration	<code>for v in l:</code>	$O(N)$	
sort	<code>l.sort()</code>	$O(N \log N)$	
multiply	<code>k*l</code>	$O(kN)$	

Python List는
동적 배열로
구현됨

Tuple

- 불변 타입 리스트 (Immutable List)
- 선언 시 "[]" 가 아닌 "(" ")"를 사용, 문맥에 따라 괄호 생략 가능
- 리스트의 연산, 인덱싱, 슬라이싱 등을 동일하게 사용

```
>>> t = (1, 2, 3, 4)                      # 선언
>>> t = 1, 2, 3, 4                         # 괄호 생략 가능
>>> t
(1, 2, 3, 4)

>>> len(t)                                 # List 메소드 함수들 사용 가능
4
>>> t * 2
(1, 2, 3, 4, 1, 2, 3, 4)

>>> t[3] = 5                                # 불변 타입이라 수정은 불가능
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Features of Tuple

- 일반적으로 함수에서 2개 이상 요소를 반환할 때 사용
- Tuple은 불변 타입 이지만 Tuple안의 요소는 가변 타입일 수도 있음
- 문자열 타입 (str)의 경우 일종의 문자 tuple로 생각 가능
→ Indexing 및 slicing이 가능

```
>>> (1)          # 그냥 괄호는 산술연산자
1
>>> (1,)        # 요소 하나를 가진 Tuple을 위해서는 ","를 적어야 함
(1,)
```

```
>>> a = (1, 2, [5, 6, 7])      # 불변 타입 안의 가변 타입
>>> a[2].append(8)            # 가변 타입 내 삽입은 문제 없음
>>> a
(1, 2, [5, 6, 7, 8])
```

```
>>> a = 'some text'        # 불변 타입
>>> a[:4]                  # Tuple의 연산자 사용가능
'some'
```

Packing & Unpacking

- 패킹: 여러 데이터를 묶기
 - Ex) $t = 1, 2, 3, 4, 5$
- 언패킹: 묶인 데이터를 풀기
 - Ex) $a, b, c, d, e = t$
 - * (Asterisk)로 남는 요소를 리스트로 남기기 가능

```
>>> t = [1, 2, 3, 4, 5]
>>> a, b, c, _, _ = t    # Unpacking
>>> c                  # “_”는 관습적으로 사용하지 않는 변수에 사용
3

>>> a, *b, c = t        # * 로 남는 부분을 리스트로 묶을 수 있음
>>> a, b, c
(1, [2, 3, 4], 5)
```

Dictionary

- 일종의 맵핑을 위한 데이터 구조
 - Key → Value 형태로 구현
 - 불변 타입으로만 이루어져 있으면 Key로 사용 가능
 - {Key1: Value1, Key2: Value2, Key3: Value3, ...} 형태로 선언
- Key → Value
1 → 'something'
(1, 2.5) → 1.5
'text' → 2

```
>>> dictionary = {  
...     1: 'something',           # 1을 'something'에 맵핑  
...     (1, 2.5): 1.5,          # 정확히는 Hashable할 경우 Key로 사용 가능  
...     'text': 2,               # 마지막 콤마는 무시  
... }  
  
>>> dictionary[1]            # 대괄호로 indexing  
'something'  
  
>>> dictionary['text']  
2  
  
>>> dictionary[1, 2.5]        # Tuple이므로 괄호 생략 가능  
1.5
```

Addition & Deletion on Dictionary

```
>>> dictionary = {}                      # 빈 딕셔너리 생성, dict()로도 가능

>>> dictionary['text'] = 1                # 요소 삽입
>>> dictionary['list'] = [5, 6, 7]       # Value는 아무 객체나 가능
>>> dictionary
{'text': 1, 'list': [5, 6, 7]}

>>> dictionary['text'] = 'Oh?'           # 키는 중복이 불가능 → 덮어 씌워짐
>>> dictionary
{'text': 'Oh?', 'list': [5, 6, 7]}

>>> del dictionary['text']              # 요소 삭제
>>> dictionary
{'list': [5, 6, 7]}

>>> len(dictionary)                   # 크기 확인
1
```

Listing Dictionary Items

```
>>> dictionary = {'한국어': 0, '영어': 1, '중국어': 2}
>>> dictionary
{'한국어': 0, '영어': 1, '중국어': 2}

>>> dictionary.items()          # 모든 key와 value를 일종의 Tuple List로 반환
dict_items([('한국어', 0), ('영어', 1), ('중국어', 2)])

>>> dictionary.keys()          # 모든 Key를 일종의 List로 반환
dict_keys(['한국어', '영어', '중국어'])
>>> dictionary.values()        # 모든 Value를 일종의 List로 반환
dict_values([0, 1, 2])

>>> 2 in dictionary.values()    # Value 안에 2가 있는지 검사
True
```

Time complexity of Dictionary

Operation	Example	Big-O	Notes
Index	<code>d[k]</code>	$O(1)$	
Store	<code>d[k] = v</code>	$O(1)$	
Length	<code>len(d)</code>	$O(1)$	
Delete	<code>del d[k]</code>	$O(1)$	
Clear	<code>d.clear()</code>	$O(1)$	<code>s = {}</code> or <code>= dict()</code> 유사
Construction	<code>dict(...)</code>	$O(N)$	
Iteration	<code>for k in d:</code>	$O(N)$	

Dictionary는 Hash로 구현: indexing 속도가 $O(1)$

Set

- Dictionary의 Key만 모여 있는 형태 → 집합형

```
>>> s = set([1, 2, 3, 'text'])          # Set 선언  
>>> s  
{1, 2, 3, 'text'}  
  
>>> s.add(4)                         # 요소 추가  
>>> s.add('text')                     # 중복된 요소는 추가하지 않음  
>>> s  
{1, 2, 3, 4, 'text'}  
  
>>> s.remove(2)                      # 요소 삭제  
>>> s.remove(99)                     # 존재하지 않는 요소는 에러 발생  
Traceback (most recent call last):  
File "<stdin>", line 1, in <module>  
KeyError: 99  
  
>>> s.discard(99)                   # 요소 삭제, 존재하지 않을 경우 무시  
>>> s  
{1, 3, 4, 'text'}  
  
>>> s.update([1, 99, None, True])    # 여러 요소 추가, 중복은 무시  
>>> s  
{1, None, 3, 4, 99, 'text'}  
>>> s.clear()                       # True는 1과 동일하기 때문에 추가되지 않음  
>>> s  
set()
```

Set Operations

- 수학적 집합 연산자가 존재

```
>>> s1 = set([1, 2, 3, 4])
>>> s2 = set([3, 4, 5, 6])

>>> s1 & s2          # 교집합
{3, 4}

>>> s1 | s2          # 합집합
{1, 2, 3, 4, 5, 6}

>>> s1 - s2          # 차집합
{1, 2}

>>> s1 ^ s2          # 배타적 합집합
{1, 2, 5, 6}
```

