CHAPTER

03

파이썬 자료구조와 함수

Python Data Structures and Functions

컴퓨터소프트웨어공학과
김대영

개 요

- 파이썬 자료구조 (Data structure)
- 함수 (Function)
- 파일IO (File IO)

- **튜플**(Tuple)
  - 1차원의 고정된 크기, 이뮤터블(immutable) 자료구조
  - 튜플의 정의

```
tup = 1, 2, 3
tup
> (1, 2, 3)
```

  - 중첩된 튜플
    - 괄호( )를 이용하여 중첩된 튜플 정의

```
nested_tup = (4, 5, 6), (7, 8)
nested_tup
> ((4, 5, 6), (7, 8))
```

- **튜플**
  - 모든 순차적인 데이터는 *tuple()*을 사용하여 튜플로 변경가능

```
tuple([5, 0, 3])
> (5, 0, 3)

tup = tuple(“sample”)
tup
> (‘s’, ‘a’, ‘m’, ‘p’, ‘l’, ‘e’)
```

  - 튜플의 각 요소는 대괄호 *[ ]*를 사용하여 접근

```
tup[0]
> ‘s’
```

- **튜플**
  - 튜플에 저장된 객체는 수정 가능
  - 그러나, 한번 생성된 슬롯의 객체는 변경할 수 없음

```
tup = tuple(["string", [1, 2], True])
tup[2] = False
> TypeError: 'tuple' object does not support item assignment
```

```
tup = tuple(["foo", [1, 2], True])
tup[1].append(3)
tup
> ('foo', [1, 2, 3], True)
```

➔ tup[1] 객체를 다른 객체로 변경할 수 없지만, 값을 추가하는 것처럼 수정할 수 있음

- **튜플**
  - 서로 다른 변수로 값 분리
    - 대입연산자 (=)를 사용하여 여러 변수에 튜블 객체를 할당하면, 각각의 변수에 튜플의 원소값이 할당됨

```
tup = (4, 5, 6)
a, b, c = tup
b
> 5
```

  - 튜플 메소드 count()
    - 튜플은 이뮤터블 자료구조로 다양한 메소드를 가지고 있지 않지만, count() 메소드를 사용하여 동일 원소의 개수를 찾을 수 있음

```
a = (1, 2, 2, 2, 3, 4, 2)
a.count(2)
> 4
```

- **리스트**(List)
  - 뮤터블 자료 구조로 생성한 객체의 변경 가능
  - 대괄호 *[ ]* 와 *list()* 함수를 통해서 정의됨

```
a_list = [2, 3, 7, None]
tup = (‘foo’, ‘bar’, ‘baz’)
b_list = list(tup)

b_list
> [‘foo’, ‘bar’, ‘baz’]

b_list[1] = ‘test’
b_list
> [‘foo’, ‘test’, ‘baz’]
```

- **리스트**(List)
  - 리스트 요소(element) 추가하기:
    - 리스트 객체의 append(), insert() 메소드를 사용하여 요소를 추가할 수 있음
    - append(): 리스트의 마지막 요소 다음에 새로운 요소를 추가
    - Insert(): 리스트의 지정된 위치에 새로운 요소를 추가

```
b_list.append('dwarf')
b_list
> ['foo', 'test', 'baz', 'dwarf']

b_list.insert(1, 'red')
b_list
> ['foo', 'red', 'test', 'baz', 'dwarf']
```

- **리스트**(List)
  - 리스트 요소 삭제하기
    - 리스트 객체의 pop(), remove() 메소드를 사용하여 요소를 추가할 수 있음
    - pop(): 지정된 위치의 요소를 리스트에서 제거함
    - remove(): 리스트의 첫번째 요소부터 리스트에서 제거됨

```
b_list.pop(2)
> 'test'
b_list
> ['foo', 'red', 'baz', 'dwarf']

b_list.remove('foo')
b_list
> ['red', 'baz', 'dwarf']
```

- **리스트**(List)
  - 리스트 객체 내부의 요소 확인하기
    - In, not 을 사용하여 요소가 리스트 객체에 존재하는지 여부를 boolean으로 리턴함

```
'dwarf' in b_list
> True

'dwarf' not in b_list
> False
```

- **리스트**(List)
  - 정렬(sort)
    - 리스트 객체의 sort() 메소드를 사용하여 요소를 정렬함
    - Sort()메소드의 옵션을 사용하여 요소들의 길이에 대한 정렬을 수행할 수 있음

```
a = [7, 2, 5, 1, 3]
a.sort()
a
> [1, 2, 3, 5, 7]
```

```
b = ['saw', 'small', 'He', 'foxes', 'six']
b.sort(key=len)
b
> ['He', 'saw', 'six', 'small', 'foxes']
```

- **리스트**(List)
  - 이진탐색(Binary search)와 bisect 모듈을 사용한 정렬
    - bisect(): 리스트 객체에 새로운 요소가 추가될 때 정렬된 리스트를 유지하기 위한 위치를 리턴함
    - insort(): 정렬된 리스트를 유지하면서 새로운 요소를 리스트 객체에 추가함

```
import bisect

c = [1, 2, 2, 2, 3, 4, 7]
bisect.bisect(c, 2)
> 4
bisect.bisect(c, 5)
> 6
bisect.insort(c, 6)
c
> [1, 2, 2, 2, 3, 4, 6, 7]
```

- **리스트**(List)
  - 슬라이싱(slicing): 리스트 객체로부터 주어진 범위의 데이터 요소를 획득하는 것
    - 시작 지점의 인덱스는 포함하지만, 종료 지점의 인덱스는 포함하지 않음
    - 슬라이싱 개수 = 종료 지점 인덱스 – 시작 지점 인덱스

```
seq = [7, 2, 3, 7, 5, 6, 0, 1]
seq[1:5]
> [2, 3, 7, 5]
```

    - 슬라이싱 영역에 데이터 할당 하기

```
seq[3:4] = [6, 3]
seq
> [7, 2, 3, 6, 3, 5, 6, 0, 1]
```

- **리스트**(List)
  - 슬라이싱의 시작 지점 인덱스와 종료 지점 인덱스는 생략 가능

```
seq = [7, 2, 3, 6, 3, 5, 6, 0, 1]
seq[:5]
> [7, 2, 3, 6, 3]
seq[3:]
> [6, 3, 5, 6, 0, 1]
```

  - 슬라이싱 위치에 대한 마이너스 인덱스는 순서로 나열된 데이터 요소의 끝에서부터의 위치를 의미함

```
seq[-4:]
> [5, 6, 0, 1]
seq[-6:-2]
> [6, 3, 5, 6]
```

- **리스트**(List)
  - 중첩된 콜론(:)은 리스트 객체의 인덱스 스텝 크기를 결정

```
seq = [7, 2, 3, 6, 3, 5, 6, 0, 1]
seq[::2]
> [7, 3, 3, 6, 1]
```

  - 스텝 크기를 -1로 지정하는 경우: 리스트 요소를 역순으로 리턴

```
seq = [7, 2, 3, 6, 3, 5, 6, 0, 1]
seq[::-1]
> [1, 0, 6, 5, 3, 6, 3, 2, 7]
```

- **내장 함수**(Embedding function)
  - enumerate() 함수: 순차 자료에 대해서 개별 요소와 인덱스를 함께 다루기 위해 사용

```
for i, value in enumerate(collection):
        # write codes for value
```

```
some_list = [‘foo’, ‘bar’, ‘baz’]
mapping = {}  # dictionary
for i, v in enumerate(some_list):
        mapping[v] = i
mapping
> {‘bar’:1, ‘baz’:2, ‘foo’:0}
```

- **내장 함수**(Embedding function)
  - sorted() 함수: 순차 자료에 대해서 정렬된 값을 리턴함

```
sorted([5, 1, 2, 4, 0, 3, 2])
> [0, 1, 2, 2, 3, 4, 5]
```

  - reversed() 함수: 리스트와 같은 순차 자료의 요소들에 대해서 역순으로 순회함

```
list(reversed(range(10))
> [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

- **내장 함수**(Embedding function)
  - zip() 함수: 서로 다른 순차 자료 쌍에 대하여 튜플의 리스트를 생성함

```
seq1 = ['foo', 'bar', 'baz']
seq2 = ['one', 'two', 'three']
zipped = zip(seq1, seq2)
list(zipped)
> [('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

- **사전**(Dictionary)
  - 키(key)와 값(value)을 가지는 해시맵(Hash-map) 구조
  - 표현: {key : value}

```
empty_dict = {}
d1 = {‘a’ : ‘some value’, ‘b’ : [1, 2, 3, 4]
d1
> {‘a’: ‘some value’, ‘b’: [1, 2, 3, 4]}
```

  - 리스트나 튜플처럼 사용할 수 있음

```
d1[7] = ‘an integer’
d1
> {‘a’: ‘some value’, ‘b’: [1, 2, 3, 4], 7: ‘an integer’}

d1[‘b’]
> [1, 2, 3, 4]
```

- **사전**(Dictionary)
  - 데이터 요소의 삭제: 사전에서 키-값 쌍의 요소를 삭제하기 위해 del 명령이나 pop() 메소드를 사용함

```
d1[5] = ‘some value’
d1
> {‘a’: ‘some value’, ‘b’: [1, 2, 3, 4], 7: ‘an integer’, 5: ‘some value’}

del d1[5]
d1
> {‘a’: ‘some value’, ‘b’: [1, 2, 3, 4], 7: ‘an integer’}

ret = d1.pop(‘b’)
ret
> [1, 2, 3, 4]

d1
> {‘a’: ‘some value’, 7: ‘an integer’}
```

- **사전**(Dictionary)
  - update() 메소드: 두 개의 사전 자료를 연결하기 위해 사용

```
d1 = {'a': 'some value', 7: ' an integer'}
d1.update({'b': 'foo', 'c': 12})

d1
> {'a': 'some value', 7: 'an integer', 'b': 'foo', 'c': 12}
```

  - 리스트와 같은 두 개의 순차 자료를 사전으로 만들기 위해 zip() 메소드를 사용함

```
mapping = {}
for key, value in zip(key_list, value_list):
        mapping[key] = value
```

- **사전**(Dictionary)
  - 사전 사용법: 키를 인덱스로 하여 키에 대응되는 값을 리턴함

```
if key in some_dict:
        value = some_dict[key]
else:
        value = default_value
```

  - get() 메소드를 사용하여 키에 대응되는 값을 리턴 할 수 있음

```
value = some_dict.get(key, default_vale)
```

- **집합**(Set)
  - 집합에서는 순서 없는 데이터 타입이 요소를 구성함
  - 사전 자료구조와 유사하지만 값(valuse)이 없고, 키(key)만 존재함
  - 표기: set() or {}

```
set([2, 2, 2, 1, 3, 3])
> {1, 2, 3}

{2, 2, 2, 1, 3, 3}
> {1, 2, 3}
```

- **집합**
  - 합집합(union)과 교집합(intersection)
    - 집합의 *union()* 메소드와 *intersection()* 메소드를 사용하여 두 집합의 합집합과 교집합 원소를 찾아냄

```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}

a.union(b)    # a | b
> {1, 2, 3, 4, 5, 6, 7, 8}
```

```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}

a.intersection(b)    # a & b
> {3, 4, 5}
```

- **집합**
  - 차집합(difference)
    - difference()를 사용하여 두 집합의 차를 구함

```
a.difference(b)                    # a - b
```

  - Subset과 Superset (부분집합)
    - issubset() 메소드와 issupperset() 메소드를 사용하여 부분집합으로 포함되거나 포함하는지를 불린 값으로 표현함

```
a_set = {1, 2, 3, 4, 5}
b_set = {1, 2, 3}

b_set.issubset(a_set)
> True
a_set.issupperset(b_set)
> True
```

- **함수**(Function)
  - 함수는 특정 기능을 나타내는 일종의 모듈로 생각할 수 있음
  - 파이썬에서 함수는 *def*로 정의하고, 함수에서 값은 반환은 *return*으로 처리함

```
def my_function(x, y, z=1.5):
        if z > 1:
                return z*(x+y)
        else:
                return z/(x+y)

ret = my_function(4, 6, 3.5)
> 35.0
ret = my_function(10, 20)
> 45.0
```

- **함수의 범위**
  - 지역 변수가 사용되는 범위는 def로 정의된 함수 영역 내
  - 파이썬 함수 블록은 들여쓰기로 구분

```
def func():
        a = []          # local variable
        for j in range(5):
                a.append(j)
```

- **함수의 범위**
  - 전역 변수는 함수 밖에서 선언
  - 함수 내에서 전역 변수 사용은 *global* 키워드를 통하여 선언 후 사용

```
a = []                  # global variable
def func():
      for j in range(5):
            a.append(j)
```

```
a = None
def bind_var_func():
      global a;      # declare global variable
      a = []
      for j in range(5):
            a.append(j)
```

- **여러 값 반환**
  - 파이썬에서는 함수에서 여러 값을 반환 받는 것을 허용

```
def f():
        a = 5
        b = 6
        c = 7
        return a, b, c

a, b, c = f()
```

- **객체로써의 함수**
  - 파이썬에서는 모든 자료와 함수는 객체로 다루어짐
  - 따라서, 함수에서 객체를 생성하고 제어하기 쉬움
  - 예) 다음의 예제처럼 states 리스트를 함수내에서 객체 단위의 메소드를 사용하여 쉽게 정형화할 수 있음

```
import re
states = [‘Alabama’, ‘Georgia!’, ‘Georgia’, ‘georgia’,
                ‘FlOrIda’, ‘southcarolina##’, ‘West virginia?’]

def clean_strings(strings):
        result = []
        for value in strings:
                value = value.strip()
                value = re.sub(‘[!#?]’, ‘’, value)
                value = value.title()
                result.append(value)
        return result
```

```
clean_strings(states)
> [‘Alabama’,
   ‘Georgia’,
   ‘Georgia’,
   ‘Georgia’,
   ‘Florida’,
   ‘South Carolina’,
   ‘West Virginia’]
```

- **람다 함수**(Lambda function)
  - 익명 함수(Anonymous function)로 알려짐
  - 리턴 값을 가지는 단순한 구조의 문장으로 구성됨
  - *lambda* 키워드로 정의되며 익명 함수 선언을 의미함

```
def short_function(x):
        return x * 2

equiv_anon = lambda x: x * 2
```

```
def short_function(x):
        return x * 2

z = 2
ret1 = short_function(z)
ret2 = (lambda x: x * 2)(z)
print(ret1, ret2)
```

- **제너레이터**(Generator)
  - 순차 데이터를 순회하기 위한 방법
  - 이터레이터 프로토콜(*Iterator protocol*)을 통하여 순회 가능한 객체를 생성

```
# return key in dictionary traversal
# python interpreter creates iterator

some_dict = {‘a’:1, ‘b’:2, ‘c’=3}
for key in some_dict:
      print(key)
```

- **제너레이터**(Generator)
  - 순회 객체를 생성하기 위한 단순한고 일반적인 방법
  - 제너레이터는 요청마다 하나씩 순서 데이터를 리턴
  - 제너레이터 사용을 위해 함수에서 *return* 대신에 *yield*를 사용

```
def squares(n=10):
        print('Generating squares from 1 to {0}'.format(n**2))
        for j in range(1, n + 1):
                yield j**2
```

- **제너레이터**(Generator)
  - 간단한 제너레이터 표현

```
gen = (x ** 2 for x in range(10))
l = list(gen)
print(l)
> [1, 4, 9, 16, 25, 36, 47, 64, 81]
```

- **예외처리**(Exception)
  - 파이썬의 예외처리는 *try-except* 구문으로 처리함

```
def attempt_float(x):
        try:
                return float(x)
        except ValueError:
                return x

ret = attempt_float('1.23')
print(ret)
> 1.23

ret = attempt_float('something')
print(ret)
> 'something'
```

- **파일IO**
  - 파이썬에서의 파일 입출력은 open(), close 함수를 사용
  - 파일 open시 파일의 절대 경로 또는 상대 경로를 인자로 넘겨줌

```
path = 'examples/address.txt'
f = open(path)                  # file open
…
f.close()                       # file close
```

- **파일IO**
  - File open: open(path, mode)

```
r               # read only
w               # write only (overwrite available)
x              # write only (if there is the same file, fail)
a               # append
r+              # read + write
b               # binary mode; ex. ‘rb’, ‘wb’
t               # text mode (default)
```

- **파일IO**
  - 파이썬은 다양한 파일 메소드를 지원하여 파일IO를 쉽게 다룸
  - 파일 객체는 다음의 메소드를 가짐

```
read(size)              # reads a file as the given size
                        #       and returns it as a string
readlines(size)         # reads a file in lines as the given size
                        #       and returns it as a list
write(str)              # write the given string to a file
writelines(strings)     # write the given strings to a file
close()                 # close file handle
flush()                 # refresh I/O buffer
seek(pos)               # move to the given position (pos)
tell()                  # return integer number for current position
```

- **유니코드 파일**(Unicode files)
  - 유니코드 파일 읽기

```
# when data is encoded with utf-8 (Non-ASCII),
with open(path, ‘rb’) as f:
        data = f.read(10)

data.decode(‘utf8’)
```

  - 파일 오픈시 인코딩 모드 변환

```
with open(path, encoding=‘iso-8859-1’) as f:
        print(f.read(10))
```

  - 문자열을 utf-8로 변환

```
str = ‘test string’
encoded = str.encode(‘utf-8’)
```

파이썬 실습

```
from datetime import datetime, date, time
import bisect
import re

def main():

    tup =4, 5, 6
    print(tup)

    n_tup = (4,5, 6), (7, 8)
    print(n_tup)

    l = [4,0,2]
    print(l)
    tup = tuple(l)
    print(tup)
    print(tup[0])

    tup = tuple(["foo", [1,2], True])
    print(tup)
    tup[2] = False
    tup[1].append(3)
    print(tup)

    tup = (4,5,6)
    a, b, c = tup
    print(b)
    print(c)

    a_list = [2,3,7,None]
    tup = ("foo", "bar", "baz")
    print(a_list)
    print(tup)
```

```
    b_list = list(tup)
    print(b_list)
    b_list[1] = "test"
    print(b_list)

    b_list.append("dwarf") # @ end of the list
    print(b_list)
    b_list.insert(2, "red") # @index 2, "read"
    print(b_list)
    print("dwarf" in b_list)
    print("dwarf" not in b_list)

    a = [7, 2, 5, 1, 3]
    a.sort()
    print(a)
    b = ["saw", "small", "He", "foxes","six"]
    b.sort(key=len)
    print(b)

    c = [1,2,2,2,3,4,7]
    ret = bisect.bisect(c,5)
    print(ret)
    bisect.insort(c, 6)
    print(c)

    seq = [7,2,3,7,5,6,0,1]
    print(seq[1:5])
    seq[3:4] = [6,3]
    print(seq)
    print(seq[:5])
    print(seq[3:])
    print(seq[-4:])
    print(seq[-6:-2])
```

파이썬 실습

```
some_list = ["foo", "bar", "baz"]
mapping = {}
for i, v in enumerate(some_list):
    mapping[v] = i
print(mapping)

l = sorted([7,1,2,6,0,3,2])
print(l)

seq1 = ["foo", "bar", "baz"]
seq2 = ["one", "two", "three"]
zipped = zip(seq1, seq2)
print(list(zipped))

l = list(reversed(range(10)))
print(l)

d1 = {'a' : 'some_value', 'b' : [1,2,3,4]}
#print(d1)

d1[7] = 'an integer'
#print(d1)
#print(d1['b'])
d1[5] = 'some value'
print(d1)
del d1[5]
print(d1)
d1.pop('b')
print(d1)

d1 = {'a':'some value', 7: 'an integer'}
d1.update({'b' : 'foo', 'c' : 12})
print(d1)
```

```
key_list = ['a', 'b','c']
value_list = ["foo", "bar", "baz"]
mapping={}
for key, value in zip(key_list, value_list):
    mapping[key] = value
print(mapping)

default = "default"
some_dict = {'a':'some value', 7: 'an integer'}
value = some_dict.get('a', default)
print(value)

s = set([2,2,2,1,3,3])
print(s)
s = {2, 2, 2, 1, 3, 3}
print(s)

a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}

# a | b
s = a.union(b)
print(s)
# a & b
s = a.intersection(b)
print(s)
# a - b
s = a.difference(b)
print(s)
```

파이썬 실습

```
def my_func(x, y, z = 1.5):
    if z > 1:
        return z*(x+y)
    else:
        return z/(x+y)
ret = my_func(4, 6, 3.5)
print(ret)
ret = my_func(10, 20)
print(ret)
ret = my_func(10, 20, -1)
print(ret)

def func():
    a = []
    for j in range(5):
        a.append(j)
    print(a)
func()
b = []
def func2():
    for j in range(5):
        b.append(j)
    print(b)
func2()
c = None
def func3():
    global c
    c = []
    for j in range(5):
        c.append(j)
    print(c)
func3()
```

```
    def f():
        a = 5
        b = 6
        c = 7
        return a, b, c

    l, m, n = f()
    print(l, m, n)

    states = ['Alabama', 'Georgia!', 'Georgia', 'georgia
',
            'FlOriDa', 'south carolina##', 'West virgini
a?']
    def clean_str(strings):
        ret = []
        for value in strings:
            value = value.strip()
            value = re.sub('[!#?]', '', value)
            value = value.title()
            ret.append(value)
        return ret

    s = clean_str(states)
    print(s)

    def short_func(x):
        return x*2
    z = 2
    ret1 = short_func(z)
    ret2 = (lambda x: x*2)(z)
    print(ret1, ret2)
```

파이썬 실습

```
    def attempt_float(x):
        try:
            return float(x)
        except ValueError:
            return x
    ret = attempt_float('1.23')
    print(ret)
    ret = attempt_float('something')
    print(ret)

if __name__ == '__main__': # main()
    main()
```