DS 부트 캠프 2024 Day1(v1.2)

한상곤(Sangkon Han)
부산대학교 정보컴퓨터공학부
2024/02/07

목차

리스트

컨테이너 자료형

- 유사한 특성을 가진 변수가 여러개 필요한 경우, 데이터가 7개 있기 때문에 7개의 개별적인 변수를 선언하여 이 변수에 값을 담아서 사용해야 함
- 개별 원소를 복사하고 조작하려면 많은 코딩이 필요 => 연속적인 자료값들은 컨테이너 변수와 인덱스를 통해서 참조하는 것이 가능
- 변수를 많이 생성하는게 왜 나쁜가?
  - 개별적인 변수의 생성과 값의 할당으로 이루어진 메모리 구조에 대한 이해도가 높아야 함
    - 많은 변수 이름을 만들어야 함
    - 일괄적인 처리가 힘듬
    - 제어문과 반복문을 사용함에 있어서 일관된 논리적인 흐름을 만들기 어려움

컨테이너 자료형의 구성 요소

- 항목^{item} 또는 요소^{element}
- 쉼표(,)를 통해서 항목 또는 요소를 구별

```
lst = [1, 2.5, "apple"]
tup = (1,2.5,"apple")
dct = {"a":"apple", "b":"best", "c": 4.5}
st = {1, 2.5, "apple", 1, 2.5}
```

리스트

- 리스트는 대괄호 `[]` 내에 쉼표를 이용하여 값을 구분
- '세 번째 변수'와 같이 위치를 지정 `index` 해서 원하는 값을 불러오는 것도 가능

```
list1 = list()                  # 빈 리스트 생성하기 1
list2 = []                      # 빈 리스트 생성하기 2

list3 = list((1, 2, 3))         # 튜플로부터 리스트 생성
list4 = list(range(1, 10))      # range() 함수로부터 리스트 생성

list5 = list('ABCDEF')          # 문자열로부터 리스트 생성
```

인덱스 index

- 항목 값을 가리키는 숫자
  - `n` 개의 항목을 가진 리스트의 인덱스는 0부터 n-1 까지 증가
- 인덱싱 `indexing`
  - 항목의 인덱스를 이용하여 자료값에 접근

```
n_list = [11, 22, 33, 44, 55, 66]
len(n_list)  # 리스트의 요소의 개수를 구하는 함수
n_list[0]    # 리스트의 첫 번째 항목의 인덱스는 0이다.
n_list[1]    # 리스트의 두 번째 항목의 인덱스는 1이다.
```

인덱스 `index` 주의사항

- 인덱스 값에 최대 인덱스 값보다 더 큰 값을 넣으면 안 됨
  - 최대 인덱스는 `len(n_list)-1`

```
n_list = [11, 22, 33, 44, 55, 66]
n_list[5]
n_list[6]
```

음수 인덱스 index

- 파이썬 리스트는 음수 인덱스를 이용하여 원소들을 참조 가능
  - 마지막 원소로부터 -1, -2, -3과 같이 -1씩 감소하면서 인덱싱

```
n_list = [11, 22, 33, 44, 55, 66]
n_list[-1]
n_list[-2]
n_list[-3]
```

슬라이싱 slicing

- 리스트내의 항목을 특정한 구간별로 선택하여 잘라내는 기능
- 구간을 명시하기 위해 리스트_이름[start : end] 문법 사용
- end-1까지(end 미만)의 항목을 새 리스트에 삽입

```
a_list = [10, 20, 30, 40, 50, 60, 70, 80]
a_list[0:1] # [10]
a_list[0:2] # [10, 20]
a_list[0:5] # [10, 20, 30, 40, 50]
# 시작/끝 인덱스 생략가능
a_list[1:] # [20, 30, 40, 50, 60, 70, 80]
a_list[:5] # [10, 20, 30, 40, 50]
```

슬라이싱 `slicing`

문법	하는일
list[s:e]	`s`부터 `e-1`까지의 항목들을 슬라이싱
list[s:]	`s`부터 리스트 끝까지
list[:e]	처음부터 `e-1`까지 슬라이싱
list[:]	리스트 전체
list[s:e:st]	`s`부터 `e-1`까지 `st`만큼 건너띄며 슬라이싱
list[-2:]	뒤에서부터 두개의 항목을 슬라이싱
list[:-2]	끝의 두 개를 제외한 모든 항목을 슬라이싱
list[::-1]	모든 항목을 역순으로 슬라이싱
list[1::-1]	처음의 두 개 항목을 역순으로 슬라이싱

추가와 삭제

- append(), 추가
- remove(), 삭제

```
a_list = ['a', 'b', 'c', 'd', 'e']
a_list.append('f') # 'f' 항목 추가
n_list = [10, 20, 30, 40]
n_list.append(50) # 50 항목 추가
```

단순 자료구조

스택, 큐, 튜플, 딕셔너리 그리고 집합

자료구조의 개념

- 자료구조(data structure)
  - 데이터의 특징을 고려하여 저장하는 방법
- 파이썬의 대표적인 자료구조
  - 스택(stack), `Last In First Out`
  - 큐(queue), `First In First Out`
  - 튜플(tuple), 리스트(`list`)와 유사하지만 데이터 변경이 불가
  - 집합(set), 수학의 집한 연산을 지원하는 구조
  - 딕셔너리(dictionary), K/V로 구성된 저장구조
- collection
  - 파이썬에서 제시된 자료구조를 효율적으로 활용하기 위한 모듈

스택

- 스택(stack)
  - 자료구조의 핵심 개념
  - 리스트와 비슷하지만 저장 순서가 바뀌는 형태
  - 푸시(push), 스택에 데이터를 저장하는 것
  - 팝(pop), 데이터를 추출하는 것

```
st = [1, 2, 3, 4, 5]
st.append(10) # [1, 2, 3, 4, 5, 10]
st.append(20) # [1, 2, 3, 4, 5, 10, 20]
st.pop() # 20
st.pop() # 10
```

큐

- 큐(queue)
  - 스택과 다르게 먼저 들어간 데이터가 먼저 나오는 'Fist in First Out (FIFO)'의 메모리 구조를 가지는 자료구조

```
st = [1, 2, 3, 4, 5]
st.append(10) # [1, 2, 3, 4, 5, 10]
st.append(20) # [1, 2, 3, 4, 5, 10, 20]
st.pop(0) # 1
st.pop(0) # 2
```

딕셔너리의 개념

- 딕셔너리(dictionary)
  - 파이썬에서 가장 많이 사용하는 자료구조
  - 영어사전에서 각 단어를 검색할 수 있도록 색인(index)을 만들어 놓고 색인을 통해 그 단어를 찾아 의미를 파악
  - 파이썬의 딕셔너리 구조에서는 데이터의 유일한 구분자인 키(key)라는 이름으로 검색할 수 있게 하고, 실제 데이터를 값(value)이라는 이름과 쌍으로 저장하여 프로그래머가 데이터를 쉽게 찾을 수 있도록 함
- 딕셔너리의 선언
  - 중괄호 `{}`를 사용하여 키와 값을 쌍으로 구성
  - `{k1:v1, k2:v2, ...}`
- 딕셔너리 연산
  - keys()

튜플

- 튜플(tuple)
  - 리스트와 같은 개념이지만 값을 변경하는 것이 불가능한 리스트로의 자료구조

```
t = (1, 2, 3)
print(t + t , t * 2, len(t))
t[1] = 5
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

집합

- 세트(set)
  - 값을 순서 없이 저장하되 중복을 불허하는 자료형

```
s = set([1, 2, 3, 1, 2, 3])
s.add(1)
s.remove(1)
s.update([1, 4, 5, 6, 7])
s.discard(3)
s.clear()
```

집합의 연산

- 합집합, 두 집합의 중복 값을 제거하고 합치는 연산
- 교집합, 두 집합 양쪽에 모두 포함된 값만 추출하는 연산
- 차집합, 앞에 있는 집합 s1의 원소 중 s2에 포함된 원소를 제거하는 연산

```
s1 = set([1, 2, 3, 4, 5])
s2 = set([3, 4, 5, 6 ,7])
s1.union(s2) # s1과 s2의 합집합 {1, 2, 3, 4, 5, 6, 7}
s1 | s2 # set([1, 2, 3, 4, 5, 6, 7]) {1, 2, 3, 4, 5, 6, 7}
s1.intersection(s2) # s1과 s2의 교집합 {3, 4, 5}
s1 & s2 # set([3, 4, 5]) {3, 4, 5}
s1.difference(s2) # s1과 s2의 차집합 {1, 2}
s1 - s2 # set([1, 2]) {1, 2}
```

collections 모듈

```
from collections import deque
from collections import OrderedDict
from collections import defaultdict
from collections import Counter
from collections import namedtuple
```

deque 모듈

- deque 모듈
  - 스택과 큐를 모두 지원하는 모듈
- deque의 사용
  - 리스트와 비슷한 형식으로 데이터를 저장해야 함

```
from collections import deque
deque_list = deque()
for i in range(5):
    deque_list.append(i)
print(deque_list) # deque([0, 1, 2, 3, 4])
deque_list.pop() # 4
deque_list.pop() # 3
deque_list.pop() # 2
deque_list # deque([0, 1])
```

deque - appendleft

- deque 모듈의 장점
  - deque는 연결 리스트의 특성을 지원함.
  - 연결 리스트(linked list): 데이터를 저장할 때 요소의 값을 한 쪽으로 연결한 후, 요소의 다음 값의 주소 값을 저장하여 데이터를 연결하는 기법
  - 연결 리스트는 다음 요소의 주소 값을 저장하므로 데이터를 원형으로 저장할 수 있음
  - 마지막 요소에 첫 번째 값의 주소를 저장시켜 해당 값을 찾아갈 수 있도록 연결시킴.

```
from collections import deque
deque_list = deque()
for i in range(5):
    deque_list.appendleft(i)
print(deque_list) # deque([4, 3, 2, 1, 0])
```

deque - rotate

```
from collections import deque
deque_list = deque()
for i in range(5):
    deque_list.appendleft(i)
print(deque_list) # deque([0, 1, 2, 3, 4])
deque_list.rotate(2)
print(deque_list) # deque([3, 4, 0, 1, 2])
deque_list.rotate(2)
print(deque_list) # deque([1, 2, 3, 4, 0])
print(deque(reversed(deque_list))) # deque([0, 4, 3, 2, 1])
deque_list.extend([5, 6, 7])
print(deque_list) # deque([1, 2, 3, 4, 0, 5, 6, 7])
deque_list.extendleft([5, 6, 7])
print(deque_list) #deque([7, 6, 5, 1, 2, 3, 4, 0, 5, 6, 7])
```

OrderedDict

- OrderedDict 모듈
  - 순서를 가진 딕셔너리 객체

```
def sort_by_key(t):
    return t[0]

from collections import OrderedDict

d = dict()
d['x'] = 100
d['y'] = 200
d['z'] = 300
d['l'] = 500

for k, v in OrderedDict(sorted(d.items(), key=sort_by_key)).items():
    print(k, v)
```

defaultdict

- 딕셔너리의 변수를 생성할 때 키에 기본값을 지정하는 방법
- 새로운 키를 생성할 때 별다른 조치 없이 새로운 값을 생성할 수 있음

```
from collections import defaultdict
d = defaultdict(lambda: 0) # Default 값을 0으로 설정
print(d["first"])

s = [('yellow',1), ('blue',2), ('yellow',3), ('blue',4), ('red',1)]

d = defaultdict(list)

for k, v in s:
    d[k].append(v)

print(d.items()) # [('blue', [2, 4]), ('red', [1]), ('yellow', [1, 3])]
```

Counter

- 시퀀스 자료형의 데이터 값의 개수를 딕셔너리 형태로 반환하는 방법

```
from collections import Counter
list_count = Counter(list("gallahad"))
print(list_count) # Counter({'a': 3, 'l': 2, 'g': 1, 'h': 1, 'd': 1})
print(c["a"]) # 3
c = Counter(a = 4, b = 2, c = 0, d = -2)
d = Counter(a = 1, b = 2, c = 3, d = 4)
print(c + d) # Counter({'a': 5, 'b': 4, 'c': 3, 'd': 2})
print(c & d) # Counter({'b': 2, 'a': 1})
print(c | d) # Counter({'a': 4, 'd': 4, 'c': 3, 'b': 2})
```

namedtuple

- 튜플의 형태로 데이터 구조체를 저장하는 방법
  - 특정 데이터의 규정된 정보를 하나의 튜플 형태로 구성해 손쉽게 사용할 수 있는 자료구조

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(11, y=22)
print(p) # Point(x=11, y=22)
print(p.x, p.y) # 11, 22
print(p[0] + p[1]) #33
```

과제03 : 병합 정렬 작성(b)

간단한 제어문 사용하여 병합 정렬 알고리즘을 구현하세요. 병합 정렬 알고리즘은 $O(nlog(n))$ 시간 복잡도 내에서 데이터를 정렬할 수 있는 효율적인 정렬 알고리즘으로, 여기서 n은 값의 개수입니다.

과제3 병합 정렬 코드 완성

- 제시된 `assignment02.py`을 사용하세요.
- 우측에 보이는 `Algorithm 1`을 참고해서 코드를 작성하세요.
- 제시된 알고리즘을 작성하시면 정렬된 결과를 확인하실 수 있습니다.

```
$ python .\assignment02.py
[2, 3, 0, 1, 5, 4, 6, 8, 7, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Algorithm 1: Merge Function

Data: A: The array to be sorted
L1: The start of the first part
R1: The end of the first part
L2: The start of the second part
R2: The end of the second part
Result: Return the merged sorted array

Function $merge(A, L1, R1, L2, R2)$**:**
    $temp \leftarrow \{\}$;
    $index \leftarrow 0$;
    while $L1 \leq R1$ ***AND*** $L2 \leq R2$ **do**
        if $A[L1] \leq A[L2]$ **then**
            $temp[index] \leftarrow A[L1]$;
            $index \leftarrow index + 1$;
            $L1 \leftarrow L1 + 1$;
        else
            $temp[index] \leftarrow A[L2]$;
            $index \leftarrow index + 1$;
            $L2 \leftarrow L2 + 1$;
        end
    end
    while $L1 \leq R1$ **do**
        $temp[index] \leftarrow A[L1]$;
        $index \leftarrow index + 1$;
        $L1 \leftarrow L1 + 1$;
    end
    while $L2 \leq R2$ **do**
        $temp[index] \leftarrow A[L2]$;
        $index \leftarrow index + 1$;
        $L2 \leftarrow L2 + 1$;
    end
    return $temp$;
end