고급통계프로그래밍 Solution 5

컴퓨터과학부 2017920054 이호준

과제 index는 번역본 기준입니다.

Ex 10.6

Solution

```
def is_sorted(seq):
    for i in range(len(seq)-1):
        if seq[i] > seq[i+1]:
            return False
    return True
```

- 인덱스를 기준으로 for문을 순회하되, 0~len(seq)-2 까지 순회를 진행한다.
- 예를들어, 길이가 5인 경우, 인덱스는 0, 1, 2, 3, 4가 있을 것이며 우리는 인덱스 0, 1, 2, 3에 대해서만 반복을 진행한다. (i+1을 참조하기 때문에)
- 만약 i번째 원소가 i+1번째 원소보다 크다면 False를 반환한다. (오름차순 기준에 어긋나기 때문)
- 모든 i에 대해 if문의 조건이 거짓이라면 오름차순을 만족한다는 의미이다. 따라서 True를 반환한다.

Result

```
# Test
print(is_sorted([1, 2, 3]))
print(is_sorted(['b', 'a']))
print(is_sorted("statistics")) # 같은 시퀀스라 가능

# in Stdout:
True
False
False
```

Ex 10.7

Solution

```
def is_anagram(seq_a, seq_b):
    if sorted(seq_a) == sorted(seq_b):
        return True
    else:
        return False
```

- 두 문자열이 애너그램이기 위한 필요충분조건은 두 문자열이 이루는 원소의 종류와 각 원소의 개수가 모두 같아야한다.
- 이를 비교하기 위해 두 문자열을 정렬한 값이 같은지 비교하였다.
- 만약 두 문자열의 정렬값이 같으면 두 문자열은 애너그램이고, 그렇지 않다면 애너그램이 아닐 것이다.

Result

```
# Test
print(is_anagram("rescue", "secure")) # anagram
print(is_anagram("apple", "banana")) # not anagram

# in Stdout:
True
False
```

Ex 11.2

Solution

```
def histogram(s):
    my_dict = {}
    for elem in s:
        my_dict[elem] = my_dict.get(elem, 0) + 1
    return my_dict
```

- if문 이하를 다음과 같이 변경하였다:
  `my_dict[elem] = my_dict.get(elem, 0) + 1`
- 이는 my_dict[elem]에 값을 assignment해주는데, `.get()` 메서드를 활용하면 KeyError가 발생할 염려 없이 elem에 해당하는 원소를 참조할 수 있다.
  - 만약에 elem key가 존재한다면 my_dict[elem] += 1와 동치이다.
  - 만약에 elem key가 존재하지 않는다면 my_dict[elem] = 1 과 같다.

Result

```
# Test
print(histogram("aaabbc"))
print(histogram("xxxxyyzzz"))

# in Stdout:
{'a': 3, 'b': 2, 'c': 1}
{'x': 4, 'y': 2, 'z': 3}
```

Ex 11.3

Solution

```
def print_hist(my_dict):
    key_set = sorted(my_dict.keys())
    for x in key_set:
        print(x, my_dict[x])
```

- Dict의 `.keys()` 함수를 이용하면 Dictionary의 key들을 리스트로 가져올 수 있다.
- 이를 기존에 활용해왔던 `sorted()` 함수를 이용하여 사전순으로 정렬을 시행한다.
- 이후 정렬된 리스트 key_set을 순회하면서 딕셔너리의 key값을 참조하면 결과적으로는 사전순으로 딕셔너리를 참조할 수 있다.

Result

```
# Test
print_hist({'a':1, 'b':2, 'f':3, 'c':4})
print_hist({"apple":3, "banana":2, "carrot":1})

# in Stdout:
a 1
b 2
c 4
f 3
apple 3
banana 2
carrot 1
```

Ex 11.4

Solution

```
def reverse_lookup(my_dict, value):
    my_key = []
    for elem in my_dict:
        if my_dict[elem] == value:
            my_key.append(elem)
    return my_key
```

- value에 해당하는 key를 담을 리스트 `my_key`를 선언한다 (변수 생성)
- my_dict를 순회하면서 my_dict[elem] == value인, 즉 value에 해당하는 key elem이 있는지를 체크하고, 만약 그렇다면 이전에 만들어둔 my_key에 append한다.
- 반복문을 종료한 후, `my_key`를 반환한다.

Result

```
# Test
print(reverse_lookup({1:1, 2:1, 3:1}, 1))
print(reverse_lookup({1:1, 2:1, 3:1}, 2))

# in Stdout:
[1, 2, 3]
[]
```