고급통계프로그래밍 Solution 4

컴퓨터과학부 2017920054 이호준

과제 index는 번역본 기준입니다.

Ex 8.4

Solution

```
def find(word, letter, start_idx):
    index = start_idx
    while index < len(word):
        if word[index] == letter:
            return index
        index = index + 1
    return -1
```

- 기존의 `find()`에 start_idx 매개변수를 추구하였다.
- 기존에는 index = 0으로 초기화했던 것에 반해, 3번째 넣어주는 인자로 index를 초기화해준다. (index = start_idx)
- 이를 통해 start_idx부터 len(word)까지 탐색을 진행할 수 있다.

Result

```
# Test
print(find("python", 'p', 0))
print(find("python", 'p', 1))

# in Stdout:
0
-1
```

Ex 8.11

Solution 1

```
def any_lowercase1(s):
    for c in s:
        if c.islower():
            return True
        else:
            return False
```

- 올바른 함수

Solution 2

```
def any_lowercase2(s):
    for c in s:
        if 'c'.islower():
            return 'True'
        else:
            return 'False'
```

- 올바르지 않은 함수
- 문자열 'c'가 lowercase인지 확인하므로, 3번째 줄의 if문의 조건은 **항상 참**이 된다.
- 반환값은 문자열 `'True'`를 반환한다. (bool형이 아님!)

Solution 3

```
def any_lowercase3(s):
    for c in s:
        flag = c.islower()
    return flag
```

- 올바르지 않은 함수
- 시퀀스 s의 원소를 순회하는 건 좋지만 flag를 항상 갱신하게 된다.
- 따라서 인덱스 len(s)-1 (s>0) 의 원소의 소문자 여부를 반환하게 된다. 즉 마지막 원소의 `.islower()`만이 영향을 미치게 된다.

Solution 4

```
def any_lowercase4(s):
    flag = False
    for c in s:
        flag = flag or c.islower()
    return flag
```

- 올바르지 않은 함수
- flag와 `c.islower()`를 OR연산하는데, `c.islower()`가 한번이라도 True가 나오면 결과적으로 flag는 true가 발생하게 된다.
- 이는 문자열 s에 소문자가 존재하는지 확인하는 코드이다.

Solution 5

```
def any_lowercase5(s):
    for c in s:
        if not c.islower():
            return False
    return True
```

- 올바른 함수

Ex 8.12

Solution

```
def rotate_word(target, offset):
    return_str = ""
    if offset % 26 == 0:
        return target
    for elem in target:
        if offset > 0:
            if (ord(elem) + offset > 122):
                current_char = chr(96 + (ord(elem) + (offset % 26) - ord('z')))
            else:
                current_char = chr(ord(elem) + offset)
        else:
            if (ord(elem) + offset < 97):
                current_char = chr(123 - (ord(elem) + (abs(offset) % 26) - ord('a')))
            else:
                current_char = chr(ord(elem) + offset)
        return_str += current_char
    return return_str
```

- 바꿀 문자열 `target` 과 움직일 변동 `offset`을 입력받는다.
  - 만약 offset이 26의 배수라면 (ex : 26, -52 등) 결과물은 target과 동일하다. 따라서 target을 바로 반환한다.
- 시퀀스 target의 원소 elem으로 순회를 진행한다.
  - 만약 offset이 양수이면서 회전시킨 결과가 'z'(122)를 초과하는 경우라면, 초과한 만큼의 실제 변량 `(ord(elem) + (offset % 26) - ord('z'))` 을 구한 후, 96을 더해준다. ('a'는 97이고, 변량 delta의 범위는 $1 \leq delta \leq 25$ 이기 때문이다.
  - 만약 offset이 음수이면서 회전시킨 결과가 'a'(97)를 밑도는 결과가 나오는 경우라면, 초과한 만큼의 실제 변량 `(ord(elem) + (abs(offset) % 26) - ord('a'))` 을 구한 후, 123에서 변량을 빼준다. ('z'는 122이고, 변량 delta의 범위는 $1 \leq delta \leq 25$ 이기 때문이다.
  - 각 원소를 변환한 값을 current_char 변수에 담고, 이를 return_str 문자열에 concat한다.
- 작업이 완료되면 회전이 완료된 문자열이 return_str이고, 이를 반환한다.

Result

```
# Test
print(rotate_word("aaa", -27))
print(rotate_word("abc", 3))
print(rotate_word("xyz", 3))
print(rotate_word("python", 100))

# in Stdout:
zzz
def
abc
lupdkj
```

Discussion

- 함수를 다 작성하고 보니 offset를 for문에 대입하기 전에 미리 % 연산을 진행했다면 (ex : -27 → -1으로) 더욱 편하게 코드를 짤 수 있었을 것이라 판단된다.