Seminar Week 1

Swapping

Swapping the values of two variables `a` and `b`, using auxiliary variables:

```
temp = a
a = b
b = temp
```

The same with a method:

```python
def swap(x,y):
    temp = x
    x = y
    y= temp

a=1
b=2

print(a,b)
swap(a,b)
print("swapped a and b")
print(a,b)
```

- **Does it work?** Why / why not?

Answer:

```
1 2
swapped a and b
1 2
```

Performing this swap with a function does not work, because when we call the function `swap(a,b)`, Python passes the *values* of the variables `a` and `b` to the variables `x` and `y` (which are local to `swap()`), and hence any change to the variables `x` and `y` does not effect the variables `a` and `b`; we also say that Python uses *call by value* for arguments like whole numbers, strings or tuples (immutable objects).

Mutable objects

An object whose internal state can be changed is called a *mutable object*. Examples of mutable objects are Lists, Sets, Dictionaries, bytes and arrays. User-defined classes can be mutable or immutable, depending on whether we can change their internal state.

WordCount

Model solution:

```python
def wordcount(text):
    count = 0
    for pos in range(0,len(text)):
        if (pos==0 or text[pos-1].isspace()) and not text[pos].isspace():
            count += 1
    return count
```

1. What is the **algorithm** used here?
2. Which algorithm did **you** use? Is it the same?
3. What about **special cases**? Why does this work if the string is empty, has only spaces, starts with spaces, or ends with spaces?

Answers:

The algorithm here is:

- Go through the string character by character, and count how many times a *new word starts*, where a new word starts when a whitespace is followed by a non-whitespace, or there is a non-whitespace at the start of the string.

Alternatively, one could instead count the number of times a word ends.

The condition in the for-loop of this method guarantees that the algorithm does not crash (and that it outputs 0) for the empty string. Further, in the condition of the if-statement, it is important that we check if `pos==0` first, because in this case we should not call `text.[pos-1]` as this would result in a runtime error. For this to work it is important that Python evaluates boolean expressions “lazily”: In the or-statement, if the first part `pos==0` is `true` then Python does not check the second part.

Here is an alternative solution that uses `while` loops.

```
def wordcount(text):
    pos = 0
    count = 0
    while pos < len(text):
        while pos < len(text) and
text[pos].isspace():
            pos += 1
        if pos < len(text):
            count += 1
            while pos < len(text) and
not(text[pos].isspace()):
                pos += 1
    return count
```