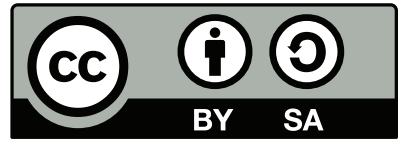
Scientific Programming

Lecture AE1 – Exercises

Andrea Passerini

Università degli Studi di Trento

2019/06/26



Exercise – Caps Lock

Alan has written a very long message for Alonzo, but at the end of writing he realized that the caps lock was active! Instead of writing the message again, Alan wants to write a Python script to change the case of letters, from upper to lower and vice-versa. Help Alan to write the algorithm!

Exercise – Caps Lock

```
tHIS IS IS A SECRET MESSAGE FROM cia.
cia IS THE cENTRAL iNTELLIGENCE aGENCY OF usa.
cia HAS 21575 EMPLOYEES.
```

```
This is a secrete message from CIA.
CIA is the Central Intelligence Agency of USA.
CIA has 21575 employees.
```

- Input: a file containing several lines of text
- Function: takes a string and convert uppercase letters to lowercase and vice-versa (implement, don't use, `string.swapcase()`)
- Output: print the converted text

Exercise – Parentheses

Donald has written a program that contains several complex expressions, composed of normal parentheses.
Unfortunately, the compiler has reported a parenthesis error, without explaining where the error is... help him by checking if the parenthesization of each expression is correct.

Exercise – Parentheses

`(((x+y)*7)-5`	`False`
`)(x+7)(`	`False`
`(3*((4+2)*8)/(4)((3*7)*14)-(3*3)`	`False`
`(x+4)(x+5)((x+5)*3)`	`True`
`(((8+2)*2)/4)((3+7)*14)-(3*3)`	`True`

- Input: a file containing several lines of text, containing text and parentheses
- Function: takes a string and returns `True` if the parentheses are correct, `False` otherwise.
- Output: print `True` or `False` for each line

Exercise – Duckburg

Fethry Duck (Paperoga) has `n` friends in Duckburg (Paperopoli). He has stored the telephone numbers of all his friends in a file, one per line. Now, he wants to discover what is the telephone prefix of Duckburg, assuming that the prefix is the longest common prefix of all numbers. Help Fethry Duck to solve the problem!

Exercise – Duckburg

```
12345
12395
12349
12312

123
```

```
919239144321
919239143321
919239124891
91923914431
919239144621
91923914202737

9192391
```

- Input: a file containing several lines of text. All lines contain a single telephone number, composed of all digits.
- Function: takes a list of numbers and return the longest string which is prefix of all of them
- Output: print the longest prefix

Exercise – Magic square

Given a file f containing n lines, each of them containing n integers separated by spaces, write a program to verify if the file contains a magic square: i.e. it contains numbers between 1 and n^2 such that each cell contains a different integer and the sum of the integers in each row, column and diagonal is equal

Example:

```
7     12     1     14
2     13     8     11
16     3     10     5
9     6     15     4
```

Exercise – Usernames

William spends most of his time administering the Italian portal for the IOI training. During his last sleepless night debugging the system, he figured out that several of the usernames used in the website are very similar. This causes a lot of confusion, especially when inspecting users' rankings. He is then considering to add a new rule to the website: you are not allowed to select a username whose set of letters is a subset of the set of letters of another username. For example, you cannot register the username `bob00` if `n0ob` is already present. In order to evaluate the impact of such a design choice, he now wants to measure how much this rule is violated by the current list of usernames.

More precisely, given a list of `n` distinct usernames `U` consisting of characters `'a' - 'z'` and `'0' - '9'`, he wants to print all the username pairs (i, j) such that $letters(U_i) \subseteq letters(U_j)$.

Exercise – Usernames

```
carole
rollercar
ndr31
xfkfk
```

```
carole < rollercar
rollercar < carole
```

```
robin
tyrionboss
bornin2000
toy
```

```
robin < tyrionboss
robin < bornin2000
toy < tyrionboss
```

- Input: a file containing several usernames, one per line of text.
- Function: takes two usernames and returns `True` if one is a subset of the other, `False` otherwise.
- Output: print all the pair for which the previous function returns `True`

Exercise – Kabbalah

George has recently enrolled in a Kabbalah seminar, where a few cultured savants debate on the concealed wisdom of the Bible. In particular, their favorite hobby is searching the Bible for long sequences of equal letters. Firstly, the seminar master selects an $N \times M$ rectangle of characters $C_{i,j}$ from the holy scriptures. Afterwards, the adepts search in all directions (horizontal, vertical and diagonal) for contiguous sequences of equal letters and the longer the sequence found, the greater the glory for the finder.

Exercise – Kabbalah

```
uieiki
zqeitz
cheuth
keinoi
```

```
aqnrb
baqah
zanqi
oahqq
rovna
zgpao
fjqhr
```

- Input: a file containing several lines of text, of equal length.
- Function: a function that, given a string, returns the length of the longest substring of equal characters.
- Function: a function that calls the previous function on all rows, columns, and diagonals
- Output: print the length of the longest (equal) substring

Spoiler alert!

Caps Lock - Solution

```
def swapcase(S):
    L=[]
    for c in S:
        if c.lower() != c:
            L.append(c.lower())
        elif c.upper() != c:
            L.append(c.upper())
        else:
            L.append(c)
    return "".join(L)

f = open("caps.txt")
for line in f:
    print(swapcase(line), end="")
```

Parentheses - Solution

```
def check(S):
    opened = 0
    for c in S:
        if c == "(":
            opened = opened+1
        elif c == ")":
            opened = opened-1
            if (opened < 0):
                return False
    return (opened == 0)

f = open("par.txt")
for line in f:
    print(check(line.strip()))
```

Duckburg - Solution

```
def prefix(numbers):
    minlen = min([len(number.strip()) for number in numbers])
    for i in range(minlen):
        col = [number[i] for number in numbers]
        if col.count(col[0]) != len(col):
            return numbers[0][:i]
    return numbers[0][:minlen]


f = open("duckburg.txt")
numbers = [number.strip() for number in f]
print(prefix(numbers))
f.close()
```

Exercise – Magic square

```
def ismagic(L):
    n = len(L)
    tot = sum(L[0])
    for i in range(n):
        val = sum(L[i])
        if val != tot:
            return False
    for j in range(n):
        val = sum([L[i][j] for i in range(n)])
        if val != tot:
            return False
    val = sum([L[i][i] for i in range(n)])
    if val != tot:
        return False
    val = sum([L[i][n-1-i] for i in range(n)])
    if val != tot:
        return False
    return True
```

Usernames - Solution

```
def included(S1,S2):
    for c in S1:
        if c not in S2:
            return False
    return True

f = open("usernames.txt")
usernames = [ line.strip() for line in f]
for user1 in usernames:
    for user2 in usernames:
        if user1 != user2 and included(user1,user2):
            print(user1,user2,sep= " < ")
f.close()
```

Usernames - Solution

```
f = open("usernames.txt")
usernames = [ line.strip() for line in f]
for user1 in usernames:
    for user2 in usernames:
        if user1 != user2 and set(user1) <= set(user2):
            print(user1,user2,sep= " < ")
f.close()
```

Kabbalah – Solution

```
# Returns the length of the longest subsequence of equal characters
def checkString(S):
    prev = ""
    count = 0
    longest = 0
    for c in S:
        if c != prev:
            count = 0
            prev = c
        count = count + 1
        longest = max(longest, count)
    return longest

f = open("kabbalah.txt")
rows = [row.strip() for row in f]
nrows = len(rows)
ncols= len(rows[0])

# Compute cols
cols = [ "".join([row[i] for row in rows])
         for i in range(ncols) ]
```

Kabbalah – Solution

```
# Compute diags
diags = []
for i in range(-ncols+1,nrows):
    diag = []
    j = 0
    for inc in range(min(ncols,nrows)):
        if (0 <= i+inc < nrows) and (0 <= j+inc < ncols):
            diag.append(rows[i+inc][j+inc])
    diags.append("".join(diag))

# Search the maximum length
maxrows = max([checkString(row) for row in rows])
maxcols = max([checkString(col) for col in cols])
maxdiag = max([checkString(diag) for diag in diags])
print(max(maxrows, maxcols, maxdiag))
f.close()
```