LAB MODULE 5

Modules, packages, string and list methods,
and exceptions

Nama : Muchamad Lutfi Maftuh
NIM : 19537141023
Prodi : Teknologi Informasi

LAB 5.1.6.4 (Reading ints safely)

Objectives

- improving the student's skills in defining functions;
- using exceptions in order to provide a safe input environment.

Scenario

Your task is to write a **function able to input integer values and to check if they are within a specified range**.

The function should:

- accept three arguments: a prompt, a low acceptable limit, and a high acceptable limit;
- if the user enters a string that is not an integer value, the function should emit the message `Error: wrong input`, and ask the user to input the value again;
- if the user enters a number which falls outside the specified range, the function should emit the message `Error: the value is not within permitted range (min..max)` and ask the user to input the value again;
- if the input value is valid, return it as a result.

Code Program

```
def readint(prompt, min, max):
    try:
        v = int(input(prompt))
        assert v >= min and v <= max
        return v
    except ValueError:
        print("Error: wrong input")
        return readint(prompt, min, max)
    except AssertionError:
        print("Error: the value is not within permitted range (min..max")
        return readint(prompt, min, max)

v = readint("Enter a number from -10 to 10: ", -10, 10)

print("The number is:", v)
```

Test data

```
Enter a number from -10 to 10: abc
Error: wrong input
Enter a number from -10 to 10: -11
Error: the value is not within permitted range (min..max
Enter a number from -10 to 10: 10
The number is: 10
```

LAB 5.1.9.18 (Your own split)

Objectives

- improving the student's skills in operating with strings;
- using built-in Python string methods.

Scenario

You already know how `split()` works. Now we want you to prove it.

Your task is to **write your own function, which behaves almost exactly like the original** `split()` **method**, i.e.:

- it should accept exactly one argument - a string;
- it should return a list of words created from the string, divided in the places where the string contains whitespaces;
- if the string is empty, the function should return an empty list;
- its name should be `mysplit()`

Code Program

```
def mysplit(strng):
    return strng.split()

print(mysplit("To be or not to be, that is the question"))
print(mysplit("To be or not to be,that is the question"))
print(mysplit("   "))
print(mysplit(" abc "))
print(mysplit(""))
```

Test data

```
['To', 'be', 'or', 'not', 'to', 'be,', 'that', 'is', 'the', 'question']
['To', 'be', 'or', 'not', 'to', 'be,that', 'is', 'the', 'question']
[]
['abc']
[]
```

LAB 5.1.10.6 (LAB: A LED DISPLAY)

Objectives

- improving the student's skills in operating with strings;
- using strings to represent non-text data.

Scenario

It's a device (sometimes electronic, sometimes mechanical) designed to present one decimal digit using a subset of seven segments. If you still don't know what it is, refer to the following Wikipedia article.

Your task is to write **a program which is able to simulate the work of a seven-display device**, although you're going to use single LEDs instead of segments.

Code Program

```
representations = {
    '0': ('###', '# #', '# #', '# #', '###'),
    '1': ('#', '#', '#', '#', '#'),
    '2': ('###', '  #', '###', '#  ', '###'),
    '3': ('###', '  #', '###', '  #', '###'),
    '4': ('# #', '# #', '###', '  #', '  #'),
    '5': ('###', '#  ', '###', '  #', '###'),
    '6': ('###', '#  ', '###', '# #', '###'),
    '7': ('###', '  #', '  #', '  #', '  #'),
    '8': ('###', '# #', '###', '# #', '###'),
    '9': ('###', '# #', '###', '  #', '###'),
    '.': ('   ', '   ', '   ', '   ', ' #'),
}

def segment7(number):
    digits = [representations[digit] for digit in number]
    for i in range(5):
        print(" ".join(segment[i] for segment in digits))

number = input("Enter number: ")
while len(number) < 0:
    number = input("Enter number: ")

segment7(number)
```

Test data

```
Enter number: 0123456789
### # ### ### # # ### ### ### ### ###
# # #   #   # # # #   #     # # # # #
# # # ### ### ### ### ###   # ### ###
# # # #     #   #   # # #   # # #   #
### # ### ###   # ### ###   # ### ###
```

LAB 5.1.11.6 (LAB: Improving the Caesar Cipher)

Objectives

- improving the student's skills in operating with strings;
- converting characters into ASCII code, and vice versa.

Scenario

You are already familiar with the Caesar cipher, and this is why we want you to improve the code we showed you recently.

The original Caesar cipher shifts each character by one: *a* becomes *b*, *z* becomes *a*, and so on. Let's make it a bit harder, and allow the shifted value to come from the range 1..25 inclusive.

Moreover, let the code preserve the letters' case (lower-case letters will remain lower-case) and all non-alphabetical characters should remain untouched.

Your task is to write a program which:

- asks the user for one line of text to encrypt;
- asks the user for a shift value (an integer number from the range 1..25 - note: you should force the user to enter a valid shift value (don't give up and don't let bad data fool you!)
- prints out the encoded text.

Code Program

```
def encrypt(text, shift):
    cipher = ''
    for c in text:
        if c.isspace() or c.isnumeric():
            cipher += c
        elif c.isupper():
            cipher += chr((ord(c) + shift - 65) % 26 + 65)
        else:
            cipher += chr((ord(c) + shift - 97) % 26 + 97)
    return cipher

text = input("Enter text: ")
shift = 0

while shift < 1 or shift > 25:
    shift = int(input("Enter shift: "))

print(encrypt(text, shift))
```

Test data

```
Enter text: maftuh
Enter shift: 0
Enter shift: 26
Enter shift: 2
ochvwj
```

LAB 5.1.11.7 (LAB: Palindromes)

Objectives

- improving the student's skills in operating with strings;
- encouraging the student to look for non-obvious solutions.

Scenario

Do you know what a palindrome is?

It's a word which look the same when read forward and backward. For example, "kayak" is a palindrome, while "loyal" is not.

Your task is to write a program which:

- asks the user for some text;
- checks whether the entered text is a palindrome, and prints result.

Code Program

```
def isPalindrome(str):
    str = str.replace(" ", "")
    str = str.lower()
    rev = str[::-1]
    return "It's a palindrome" if str == rev else "It's not a palindrome"

str = input("Enter text: ")
print(isPalindrome(str))
```

Test data

```
Enter text: kayak
It's a palindrome
```

LAB 5.1.11.8 (LAB: Anagrams)

Objectives

- improving the student's skills in operating with strings;
- converting strings into lists, and vice versa.

Scenario

An anagram is a new word formed by rearranging the letters of a word, using all the original letters exactly once. For example, the phrases "rail safety" and "fairy tales" are anagrams, while "I am" and "You are" are not.

Your task is to write a program which:

- asks the user for two separate texts;
- checks whether, the entered texts are anagrams and prints the result.

Code Program

```
def anagram(str1,str2):
    if str1 == '' or str2 == '':
      return "Not anagram"

    str1 = str1.lower()
    str2 = str2.lower()

    list1 = list(str1)
    list1.sort()

    list2 = list(str2)
    list2.sort()

    return "Anagram" if list1 == list2 else "Not anagram"

str1 = input("string 1 : ")
str2 = input("string 2 : ")
print(anagram(str1,str2))
```

Test data

```
string 1 : Listen
string 2 : Silent
Anagram

string 1 : modern
string 2 : norman
Not anagram
```

LAB 5.1.11.9 (LAB: The Digit of Life)

Objectives

- improving the student's skills in operating with strings;
- converting integers into strings, and vice versa.

Scenario

Some say that the *Digit of Life* is a digit evaluated using somebody's birthday. It's simple - you just need to sum all the digits of the date. If the result contains more than one digit, you have to repeat the addition until you get exactly one digit. For example:

- 1 January 2017 = 2017 01 01

- 2 + 0 + 1 + 7 + 0 + 1 + 0 + 1 = 12
- 1 + 2 = 3

3 is the digit we searched for and found.

Your task is to write a program which:

- asks the user her/his birthday (in the format YYYYMMDD, or YYYYDDMM, or MMDDYYYY - actually, the order of the digits doesn't matter)
- outputs the *Digit of Life* for the date.

Code Program

```
def digitOfLife(birth):
    digit = 0
    while len(birth) > 1:
        for c in birth:
            digit += int(c)
        birth = str(digit)
        digit = 0
    return birth

birth = input('Enter birth: ')
print(digitOfLife(birth))
```

Test data

```
Enter birth: 18112001
5
```

LAB 5.1.11.10 (LAB: Find a Word)

Objectives

- improving the student's skills in operating with strings;
- using the `find()` method for searching strings.

Scenario

Let's play a game. We will give you two strings: one being a word (e.g., "dog") and the second being a combination of any characters.

Your task is to write a program which answers the following question: **are the characters comprising the first string hidden inside the second string?**

For example:

- if the second string is given as `"vcxzxduybfdsobywuefgas"`, the answer is `yes`;
- if the second string is `"vcxzxdcybfdstbywuefsas"`, the answer is `no` (as there are neither the letters "d", "o", or "g", in this order)

Code Program

```
def findWord(str1, str2):
    for c in str1:
        if str2.find(c) < 0:
            return "No"
    return "Yes"

str1 = input("String 1: ")
str2 = input("String 2: ")
print(findWord(str1,str2))
```

Test data

```
String 1: donor
String 2: Nabucodonosor
Yes

String 1: donut
String 2: Nabucodonosor
No
```

LAB 5.1.1.11 (LAB: Sudoku)

Objectives

- improving the student's skills in operating with strings and lists;
- converting strings into lists.

Scenario

As you probably know, *Sudoku* is a number-placing puzzle played on a 9x9 board. The player has to fill the board in a very specific way:

- each row of the board must contain all digits from 0 to 9 (the order doesn't matter)
- each column of the board must contain all digits from 0 to 9 (again, the order doesn't matter)
- each of the nine 3x3 "tiles" (we will name them "sub-squares") of the table must contain all digits from 0 to 9.

If you need more details, you can find them here.

Your task is to write a program which:

- reads 9 rows of the Sudoku, each containing 9 digits (check carefully if the data entered are valid)
- outputs `Yes` if the Sudoku is valid, and `No` otherwise.

Code Program

```
def createBoard():
    board = [[-1 for i in range(9)] for j in range(9)]
    for i in range(9):
        print('Line', i + 1, end='')
        line = input(': ')
        for j in range(9):
            board[i][j] = int(line[j])
    return board

def checkSudoku(board):
    # Validate line
    for line in board:
      if not checkValidLine(line):
        return "No"

    # Validate row
    for i in range(9):
      row = [-1 for x in range(9)]
      for j in range(9):
        row[j] = board[j][i]
      if not checkValidRow(row):
        return "No"

    # Validate square
    for i in range(9):
      square = [-1 for x in range(9)]
      for j in range(9):
        square[j] = board[j//3][j%3]
      if not checkValidSquare(square):
        return "No"

    return "Yes"

def checkValidLine(line):
    lineSet = set(line)
    line = list(lineSet)
    return len(line) == 9

def checkValidRow(row):
    rowSet = set(row)
    row = list(rowSet)
    return len(row) == 9

def checkValidSquare(square):
    squareSet = set(square)
    square = list(squareSet)
    return len(square) == 9
```

```
board = createBoard()
print(checkSudoku(board))
```

Test data

```
Line 1: 295743861
Line 2: 431865927
Line 3: 876192543
Line 4: 387459216
Line 5: 612387495
Line 6: 549216738
Line 7: 763524189
Line 8: 928671354
Line 9: 154938672
Yes

Line 1: 195743862
Line 2: 431865927
Line 3: 876192543
Line 4: 387459216
Line 5: 612387495
Line 6: 549216738
Line 7: 763524189
Line 8: 928671354
Line 9: 254938671
No
```