Lists

Most of programs work not only with variables. They also use lists of variables. For example, a program can handle an information about students in a class by reading the list of students from the keyboard or from a file. A change in the number of students in the class must not require modification of the program source code.

Previously we have already faced the task of processing elements of a sequence — for example, when computing the largest element of the sequence. But we haven't kept the whole sequence in computer's memory. However, in many problems it is necessary to keep the entire sequence, like if we had to print out all the elements of a sequence in ascending order ("sort a sequence").

To store such data, in Python you can use the data structure called list (in most programming languages the different term is used — “array”). A list is a sequence of elements numbered from 0, just as characters in the string. The list can be set manually by enumerating of the elements the list in square brackets, like here:

Primes = [2, 3, 5, 7, 11, 13]

Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']

The list `Primes` has 6 elements, namely: `Primes[0] == 2`, `Primes[1] == 3`, `Primes[2] == 5`, `Primes[3] == 7`,`Primes[4] == 11`, `Primes[5] == 13`. The list `Rainbow` has 7 elements, each of which is the string.

Like the characters in the string, the list elements can also have negative index, for example, `Primes[-1] == 13`,`Primes[-6] == 2`. The negative index means we start at the last element and go left when reading a list.

You can obtain the number of elements in a list with the function `len` (meaning *length of the list*), e.g.`len(Primes) == 6`.

Unlike strings, the elements of a list are changeable; they can be changed by assigning them new values.

Rainbow = ['Red', 'Orange', 'Yellow', 'Green', 'Blue', 'Indigo', 'Violet']

print(Rainbow[0])

Rainbow[0] = 'red'

```
print('Print the rainbow')

for i in range(len(Rainbow)):

    print(Rainbow[i])
```

Consider several ways of creating and reading lists. First of all, you can create an empty list (the list with no items, its length is 0), and you can add items to the end of your list using `append`. For example, suppose the program receives the number of elements in the list `n`, and then `n` elements of the list one by one each at the separate line. Here is an example of input data in this format:

```
5

1809

1854

1860

1891

1925
```

In this case, you can organize the reading from such list as follows:

```
a = [] # start an empty list

n = int(input()) # read number of element in the list

for i in range(n):

    new_element = int(input()) # read next element

    a.append(new_element) # add it to the list

    # the last two lines could be replaced by one:

    # a.append(int(input()))

print(a)
```

In the demonstrated example the empty list is created, then the number of elements is read, then you read the list items line by line and append to the end. The same thing can be done, saving the variable `n`:

```
a = []
```

```
for i in range(int(input())):
    a.append(int(input()))
print(a)
```

There are several operations defined for lists: list concatenation (addition of lists, i.e. "gluing" one list to another) and repetition (multiplying a list by a number). For example:

```
a = [1, 2, 3]
b = [4, 5]
c = a + b
d = b * 3
print([7, 8] + [9])
print([0, 1] * 3)
```

The resulting list `c` will be equal to `[1, 2, 3, 4, 5]`, and a list of `d` will be equal to `[4, 5, 4, 5, 4, 5]`. This allows you to organize the process of reading lists differently: first, consider the size of the list and create a list from the desired number of elements, then loop through the variable `i` starting with number 0 and inside the loop read `i`-th element of the list:

```
a = [0] * int(input())
for i in range(len(a)):
    a[i] = int(input())
```

You can print elements of a list `a` with `print(a)`; this displays the list items surrounded by square brackets and separated by commas. In general, this is inconvenient; in common, you are about to print all the elements in one line or one item per line. Here are two examples of that, using other forms of loop:

```
a = [1, 2, 3, 4, 5]
for i in range(len(a)):
    print(a[i])
```

Here the index `i` is changed, then the element `a[i]` is displayed.

```
a = [1, 2, 3, 4, 5]
for elem in a:
    print(elem, end=' ')
```

In this example, the list items are displayed in one line separated by spaces, and it's not the index that is changed but rather the value of the variable itself (for example, in the loop `for elem in ['red', 'green', 'blue']` variable `elem`will take the values `'red'`, `'green'`, `'blue'` successively.

Pay special attention to the last example! A very important part of Python ideology is `for` loop, which provides the convenient way to iterate over all elements of some sequence. This is where Python differs from Pascal, where you have to iterate over elements' *indexes*, but not over the elements themselves.

Sequences in Python are strings, lists, values of the function `range()` (these are not lists), and some other objects.

Here's an example showing the use of the `for` loop when you are needed to extract all the digits from a string and to make numeric list of them.

```
# given: s = 'ab12c59p7dq'

# you need to extract digits from the list s

# to make it so:

# digits == [1, 2, 5, 9, 7]

s = 'ab12c59p7dq'

digits = []

for symbol in s:

    if '1234567890'.find(symbol) != -1:

        digits.append(int(symbol))

print(digits)
```

Split and join methods

List items can be given in one line separated by a character; in this case, the entire list can be read using `input()`. You can then use a string method `split()`, which returns a list of strings resulting after cutting the initial string by spaces. Example:

```
# the input is a string

# 1 2 3

s = input() # s == '1 2 3'

a = s.split() # a == ['1', '2', '3']

print(a)
```

If you run this program with the input data of `1 2 3`, the list `a` will be equal to `['1', '2', '3']`. Please note that the list will consist of strings, not of numbers. If you want to get the list of numbers, you have to convert the list items one by one to integers:

```
a = input().split()

for i in range(len(a)):

    a[i] = int(a[i])

print(a)
```

Using the special magic of Python — generators — the same can be done in one line:

```
a = [int(s) for s in input().split()]

print(a)
```

(We will explain of how does this code work in the next section.) If you want to read a list of real numbers, you have to change the type `int` to `float`.

The method `split()` has an optional parameter that determines which string will be used as the separator between list items. For example, calling the method `split('.')` returns the list obtained by splitting the initial string where the character `'.'` is encountered:

```
a = '192.168.0.1'.split('.')

print(a)
```

In Python, you can display a list of strings using one-line commands. For that, the method `join` is used; this method has one parameter: a list of strings. It returns the string obtained by concatenation of the elements given, and the separator is inserted between the elements of the list; this separator is equal to the string on which is the method applied. We know that you didn't understand the previous sentence from the first time. :) Look at the examples:

```
a = ['red', 'green', 'blue']
```

```
print(' '.join(a))

# return red green blue

print(''.join(a))

# return redgreenblue

print('***'.join(a))

# returns red***green***blue
```

If a list consists of numbers, you have to use the dark magic of generators. Here's how you can print out the elements of a list, separated by spaces:

```
a = [1, 2, 3]

print(' '.join([str(i) for i in a]))

# the next line causes a type error,

# as join() can only concatenate strs

# print(' '.join(a))
```

However, if you are not a fan of dark magic, you can achieve the same effect using the loop `for`.

Generators

To create a list filled with identical items, you can use the repetition of list, for example:

```
n = 5

a = [0] * n

print(a)
```

To create more complicated lists you can use *generators*: the expressions allowing to fill a list according to a formula. The general form of a generator is as follows:

```
[expression for variable in sequence]
```

where *`variable`* is the ID of some variable, *`sequence`* is a sequence of values, which takes the variable (this can be a list, a string, or an object obtained using the function `range`), *`expression`* — some expression, usually depending on the variable used in the generator. The list elements will be filled according to this expression.

Here are some examples of people using generators.

This is how you create a list of `n` zeros using the generator:

```
a = [0 for i in range(5)]

print(a)
```

Here's how you create a list filled with squares of integers:

```
n = 5

a = [i ** 2 for i in range(n)]

print(a)
```

If you need to fill out a list of squares of numbers from 1 to `n`, you can change the settings of `range` to`range(1, n + 1)`:

```
n = 5

a = [i ** 2 for i in range(1, n + 1)]

print(a)
```

Here's how you can get a list filled with random numbers from 1 to 9 (using `randrange` from the module `random`):

```
from random import randrange

n = 10

a = [randrange(1, 10) for i in range(n)]

print(a)
```

And in this example the list will consist of lines read from standard input: first, you need to enter the number of elements of the list (this value will be used as an argument of the function `range`), second — that number of strings:

```
a = [input() for i in range(int(input()))]

print(a)
```

4. Slices

With lists and strings, you can do slices. Namely:

`A[i:j]` slice `j-i` elements `A[i]`, `A[i+1]`, ..., `A[j-1]`.

`A[i:j:-1]` slice `i-j` elements `A[i]`, `A[i-1]`, ..., `A[j+1]` (that is, changing the order of the elements).

`A[i:j:k]` cut with the step `k`: `A[i]`, `A[i+k]`, `A[i+2*k]`,... . If the value of `k`<0, the elements come in the opposite order.

Each of the numbers `i` or `j` may be missing, what means “the beginning of line” or “the end of line"

Lists, unlike strings, are **mutable objects**: you can assign a list item to a new value. Moreover, it is possible to change entire slices. For example:

A = [1, 2, 3, 4, 5]

A[2:4] = [7, 8, 9]

print(A)

Here we received a list `[1, 2, 3, 4, 5]`, and then try to replace the two elements of the slice `A[2:4]` with a new list of three elements. The resulting list is as follows: `[1, 2, 7, 8, 9, 5]`.

A = [1, 2, 3, 4, 5, 6, 7]

A[::-2] = [10, 20, 30, 40]

print(A)

And here, the resulting list will be `[40, 2, 30, 4, 20, 6, 10]`. The reason is, `A[::-2]` is a list of elements`A[-1]`, `A[-3]`, `A[-5]`, `A[-7]`, and that elements are assigned to 10, 20, 30, 40, respectively.

If a discontinuous slice (i.e. a slice with a step `k`, `k > 1`) is assigned a new value, then the number of elements in the old and new slices necessarily coincide, otherwise error `ValueError` occurs.

Note that `A[i]` is a list item, not a slice!

5. Operations on lists

You can easily do many different operations with lists.

x in A	Check whether an item in the list. Returns True or False
x not in A	The same as not(x in A)
min(A)	The smallest element of list
max(A)	The largest element in the list
A.index(x)	The index of the first occurrence of element x in the list; in its absence generates an exception ValueError
A.count(x)	The number of occurrences of element x in the list

Problem «Even indices» (Easy)

Statement

Given a list of numbers, find and print all the list elements with an even index number. (i.e. `A[0]`, `A[2]`, `A[4]`, ...).

Your solution

```
s=input()

a = s.split(' ')

n = len(a)

for i in range(0, n, 2):

    print(a[i], end=' ')
```

Suggested solution

```
a = input().split()
```

```
for i in range(0, len(a), 2):
    print(a[i])
```

Problem «Even elements» (Easy)

Statement

Given a list of numbers, find and print all elements that are an even number. In this case use a for-loop that iterates over the list, and not over its indices! That is, don't use `range()`

Your solution

```
a = [int(s) for s in input().split()]
for elem in a:
    if elem % 2==0:
        print(elem, end = ' ' )
```

Problem «Greater than previous» (Easy)

Statement

Given a list of numbers, find and print all the elements that are greater than the previous element.

Your solution

```
a = [int(i) for i in input().split()]
for i in range(1, len(a)):
    if a[i] > a[i - 1]:
        print(a[i], end = ' ')
```

Problem «Neighbors of the same sign» (Easy)

Statement

Given a list of numbers, find and print the first adjacent elemets which have the same sign. If there is no such pair, leave the output blank.

Your solution

```
a = [int(i) for i in input().split()]

b= []

for i in range(1, len(a)):

    if a[i]*a[i - 1] > 0:

        print((a[i-1]),(a[i]))

        break
```

Suggested solution

```
a = [int(i) for i in input().split()]

for i in range(1, len(a)):

    if a[i - 1] * a[i] > 0:

        print(a[i - 1], a[i])

        break
```

Problem «Greater than neighbours» (Easy)

Statement

Given a list of numbers, determine and print the quantity of elemets that are greater than both of their neighbors.

The first and the last items of the list shouldn't be considered because they don't have two neighbors.

Your solution

```
a = [int(i) for i in input().split()]
```

```
count = 0
for i in range(1, len(a)-1):
    if a[i] > a[i+1] and a[i] > a[i-1]:
        count += 1
print(count)
```

Problem «The largest element» (Easy)

Statement

Given a list of numbers. Determine the element in the list with the largest value. Print the value of the largest element and then the index number. If the highest element is not unique, print the index of the first instance.

Your solution

```
a = [int(i) for i in input().split()]
m = max(a)
for i in range(len(a)):
    if a[i] == m:
        print(m, i)
        break
```

Suggested solution

```
index_of_max = 0
a = [int(i) for i in input().split()]
for i in range(1, len(a)):
    if a[i] > a[index_of_max]:
        index_of_max = i
print(a[index_of_max], index_of_max)
```

Problem «The number of distinct elements» (Medium)

Statement

Given a list of numbers with all of its elements sorted in ascending order, determine and print the quantity of distinct elements in it.

Your solution

```
a = [int(s) for s in input().split()]

sum=1

for i in range(1, len(a)):

    if a[i] != a[i - 1]:

        sum+=1

print(sum)
```

Suggested solution

```
a = [int(i) for i in input().split()]

num_distinct = 1

for i in range(0, len(a) - 1):

    if a[i] != a[i + 1]:

        num_distinct += 1

print(num_distinct)
```

Problem «Swap neighbours» (Medium)

Statement

Given a list of numbers, swap adjacent items in pairs (`A[0]` with `A[1]`, `A[2]` with `A[3]`, etc.). Print the resulting list. If a list has an odd number of elements, leave the last element in place.

Your solution

```
a = [int(s) for s in input().split()]

b = []

n=len(a)

for i in range(0,n//2):

    b.append(a[2*i+1])

    b.append(a[2*i])

if n % 2 !=0:

    b.append(a[n-1])

for elem in b:

    print(elem, end = ' ')

#a = [int(i) for i in input().split()]

#for i in range(1, len(a), 2):

#    a[i - 1], a[i] = a[i], a[i - 1]

#print(' '.join([str(i) for i in a]))
```

Suggested solution

```
a = [int(i) for i in input().split()]

for i in range(1, len(a), 2):

    a[i - 1], a[i] = a[i], a[i - 1]

print(' '.join([str(i) for i in a]))
```

Problem «Swap min and max» (Medium)

Statement

Given a list of unique numbers, swap the minimal and maximal elements of this list. Print the resulting list.

Your solution

```
a = [int(s) for s in input().split()]

maxi= max(a)

mini= min(a)

for i in range(0,len(a)):

    if a[i] == maxi:

        a[i] = mini

    elif a[i] == mini:

        a[i] = maxi

for elem in a:

    print(elem, end= ' ')
```

Suggested solution

```
a = [int(s) for s in input().split()]

index_of_min = 0

index_of_max = 0

for i in range(1, len(a)):

    if a[i] > a[index_of_max]:

        index_of_max = i

    if a[i] < a[index_of_min]:

        index_of_min = i

a[index_of_min], a[index_of_max] = a[index_of_max], a[index_of_min]
```

```
print(' '.join([str(i) for i in a]))
```

Problem «The number of pairs of equal» (Hard)

Statement

Given a list of numbers, count how many element pairs have the same value (are equal). Any two elements that are equal to each other should be counted exactly once.

Your solution

```
a = [ int(s) for s in input().split()]

n = len(a)

pairs = 0

for i in range(n):

    for j in range(i + 1, n):

        if a[i]==a[j]:

            pairs+=1

print(pairs)
```

Suggested solution

```
a = [int(s) for s in input().split()]

counter = 0

for i in range(len(a)):

    for j in range(i + 1, len(a)):

        if a[i] == a[j]:

            counter += 1

print(counter)
```

Problem «Unique elements» (Hard)

Statement

Given a list of numbers, find and print the elements that appear in the list only once. The elements must be printed in the order in which they occur in the original list.

Your solution

```
a = [int(s) for s in input().split()]

b = []

for i in range(0, len(a)):

    t =  a.count(a[i])

    if t == 1:

        b.append(a[i])

for elem in b:

    print(elem, end = ' ')

a = [int(s) for s in input().split()]

b = []

for i in range(0, len(a)):

    t =  a.count(a[i])

    if t == 1:

        b.append(a[i])

for elem in b:

    print(elem, end = ' ')

a = [int(s) for s in input().split()]

for i in range(len(a)):

    for j in range(len(a)):

        if i != j and a[i] == a[j]:
```

```
        break
    else:
        print(a[i], end=' ')
```

Problem «Queens» (Hard)

Statement

In chess it is known that it is possible to place 8 queens on an 8×8 chess board such that none of them can attack another. Given a placement of 8 queens on the board, determine if there is a pair of queens that can attach each other on the next move. Print the word `NO` if no queen can attack another, otherwise print `YES`. The input consists of eight coordinate pairs, one pair per line, with each pair giving the position of a queen on a standard chess board with rows and columns numbered starting at 1.

Your solution

```
n = 8
x = []
y = []
for i in range(n):
    new_x, new_y = [int(s) for s in input().split()]
    x.append(new_x)
    y.append(new_y)
correct = True
for i in range(n):
    for j in range(i + 1, n):
        if x[i] == x[j] or y[i] == y[j] or abs(x[i] - x[j]) == abs(y[i] - y[j]):
            correct = False
if correct:
    print('NO')
else:
    print('YES')
```

Suggested solution

```
n = 8
x = []
y = []
for i in range(n):
    new_x, new_y = [int(s) for s in input().split()]
    x.append(new_x)
    y.append(new_y)
correct = True
for i in range(n):
    for j in range(i + 1, n):
        if x[i] == x[j] or y[i] == y[j] or abs(x[i] - x[j]) == abs(y[i] - y[j]):
            correct = False
if correct:
    print('NO')
else:
    print('YES')
```

Problem «The bowling alley» (Hard)

Statement

In bowling, the player starts wtih 10 pins at the far end of a lane. The object is to knock all the pins down. For this exercise, the number of pins and balls will vary. Given the number of pins *N* and then the number of balls *K* to be rolled, followed by *K* pairs of numbers (one for each ball rolled), determine which pins remain standing after all the balls have been rolled. The balls are numbered from 1 to *N* (inclusive) for this situation. The subsequent number pairs, one for each *K* represent the start to stop (inclusive) positions of the pins that were knocked down with each role. Print a sequence of *N* characters, where `"I"` represents a pin left standing and `"."` represents a pin knocked down.

Your solution

```
b = [int(s) for s in input().split()]

N = b[0]

K = b[1]

x = []

y = []

for i in range(K):

    new_x, new_y = [int(s) for s in input().split()]

    x.append(new_x)

    y.append(new_y)

a = ["I" for i in range(N)]

for i in range(0, K):

    n = y[i]-x[i]

    c = x[i]-1

    a[c:y[i]] = "."*(n+1)

print(''.join(a))
```

Suggested solution

```
n, k = [int(s) for s in input().split()]

bahn = ['I'] * n

for i in range(k):
    left, right = [int(s) for s in input().split()]
    for j in range(left - 1, right):
        bahn[j] = '.'
print(''.join(bahn))
```