UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
School of Computer Science & Applied Mathematics

Iteration Lab

1 Prerequisites

This lab assumes certain knowledge on your part. Ensure that you are comfortable with creating source code files, compiling them and running them in the terminal, as well as submitting them to the online marker. Knowledge of the basic terminal commands is also assumed.

You should be familiar with the concept of variables — the notion of type, how to declare variables, and how to assign values to them. Knowledge of Python's basic input/output mechanisms is also required, as are the branching constructs (`if/else`, `if/else if/else`, etc).

If you are unsure of any of the above, please refer to the lecture slides on Moodle or consult one of the previous labs for explanations/examples.

Again, please read this lab (and all labs) very carefully. They will contain information about how to successfully complete a given exercise, as well as some content that may not be covered during lectures.

2 Loops in Python

Loops are used in programming to repeatedly execute a block of statements. This is extremely useful, and one of the things that make computers so powerful. In these labs you will use loops for many purposes, one of which will be reading in multiple sets of input. In the previous labs, your code was only able to take in one set of input and print output once. You could copy your code and paste it multiple times so that you could make it do the same thing over and over again, but this is generally a bad thing to do for many reasons. Instead, loops could be used.

In Python, there are two types of loops: `for loops` and `while loops`.

2.1 For Loop

In Python, the general form of the `for loop` looks like this:

```
for <variable> in <list>:
    <statement to execute each time the loop runs>
```

Every time the first line of the loop executes, it assigns the next element of the list to the variable you specified, starting with the first one and working it's way to the last element. Here's a quick example:

An example of a simple `for loop` is given below.

```
sum = 0
# going to enter 8 numbers
for i in range(8):
    # i will go from 0 to 7 (we don't include 8)
    num = float(input())
    sum = sum + num
print("The sum of your numbers is", sum)
```

2.2 While Loop

The `while loop` is similar to the `for loop`, but does not have an explicit initialisation or variable update section. The form of the `while loop` looks like this:

```
while <condition>:
    <code to execute while the condition holds>
```

Again, the condition is any kind of boolean expression (of any complexity), and as long as it is true, the loop body will execute. An example of a `while loop` is given below.

```
sum = 0;
print("Enter some numbers. Type -1 to stop")
x = float(input())
while x != -1:
    sum = sum + x
    x = float(input())
print("The sum of your numbers is ", sum)
```

Make sure you understand this example in particular, as you'll need it to complete today's submissions.

Submission 1: Find the Min

In Lab 2, you wrote a program that found the minimum of 3 numbers. Your task here is to write a program that accepts many integers as input, and outputs the smallest one.

Input

The input consists of a series of *integers*, one per line. You can assume that there will always be at least one number given. The end of the input will be signalled by the number -1. When you receive a -1, your program should then output the smallest number it has seen so far. Note that -1 should be completely ignored and so should not be taken into consideration when calculating the smallest number.

Output

Print out the minimum value of the numbers.

Sample Input

```
2
3
4
5
6
7
1
-1
```

Sample Output

```
1
```

Submission 2: Calculate the Mean

Write a program that accepts many *real numbers* as input, and outputs their average.

Input

The input consists of a series of numbers, one per line. You can assume that there will always be at least one number given. The end of the input will be signalled by the number -1. When you receive a -1, your program should then output the average of all numbers received. Note that -1 should be completely ignored and so should not be taken into consideration when calculating the average.

Output

Print out the mean of the numbers.

Sample Input

```
1.2
1.9
2
5
0
-1
```

Sample Output

```
2.02
```

3 String Handling

The section provides an in-depth look at the concept of *strings*. Please read this section carefully, as many tasks and assignments will require you to be able to manipulate and generally work with strings.

3.1 Characters

Before we look at strings, let's first discuss characters: the individual units that make up strings. Each character has its own integer representation, known as it's ASCII code. This means we can easily treat characters as integers, applying arithmetic and comparison operators to them as you would an integer. The ASCII codes for all characters are available at `http://www.asciitable.com`. To illustrate, see the following code:

```
x = 65;
c1 = chr(x)  # get the character representation
print("The character value of", x, "is", c1)
c2 = 'a'
c3 = ord(c2) + 1; # ord gets the integer representation. Then add 1
print("The character after", c2, "is", chr(c3))  # convert c3 to character
print("The integer value of", chr(c3), "is", c3)
```

which produces the following output:

```
The character value of 65 is A
The character after a is b
The integer value of b is 98
```

There are also a special class of characters that correspond to characters that are neither digits, punctuation or letters. These require an *escape sequence* to represent, an example of which is the newline character `"\n"`. Here is a list of some escape sequences commonly used in `cout` statements:

Characters	Name	Function
\n	Newline	Move cursor to beginning of next line
\t	Horizontal Tab	Move cursor to next tab stop
\r	Carriage Return	Move cursor to beginning of current line
\a	Alert	Sound the bell
\\	Backslash	Print a single backslash
\"	Double Quote	Print a double quote

3.2 Introduction to Strings

Strings are a series of characters that constitute text. We can think of strings as lists of characters. In Python, string literals are represented by single or double quotation marks

3.3 Operators

Much like built-in types, we can apply appropriate operators to two or more strings. For instance, we can assign a string just as we would an integer with the = operator, and we can add (concatenate) two strings with the + operator. Furthermore, we can also apply various logical operators: we can test for equality, and check if one string is greater than another (alphabetically).

The following illustrates some of these operators being applied to strings:

```
s1 = ""
if s1 == "":
    print("s1 is empty")
s2 = "hello ";
s3 = "world";
result = s2 + s3;
print("Concatenating s2 and s3 gives ", result)
if s2 == s3:
    print("s2 and s3 are equal")
elif s2 > s3:
    print("s2 is greater than s3")
else:
    print("s3 is greater than s2")
```

and produces the following output:

```
s1 is empty
Concatenating s2 and s3 gives hello world
s3 is greater than s2
```

Question: how does the comparison of two strings work? Does it have anything to do with the ASCII value of their characters?

3.4 Basic Functionality

Indexing

Because strings are simply a sequence of characters, we can access any character of a given string at a specific index (position). This is done using the notation `<string>[<index>]`. For instance, if we wish to access the first and last character of a string, we'd do the following:

```
s = "Hello";
firstChar = s[0]  # We start indexing from 0; firstChar is now 'H'
lastChar = s[-1]  # The last index is one less than the length of s
```

Iterating

We can also loop over each character of a string. We can loop over each character directly, as in the following code.

```
print("Enter a string")
s = input()
length = len(s)
for i in range(length):
    print("The character at position", i, "is", s[i])
```

Alternatively, we can loop from 0 to the length of the string. We make use of the `len()` function that returns an integer specifying the number of characters in a string. The following example loops over each character in a string and outputs the character along with its index:

```
print("Enter a string")
s = input()
for i in range(len(s)):
    print("The character at position", i, "is", s[i])
```

Submission 3: Upshift

In the film *2001: A Space Odyssey*, the artificially intelligent computer is given the name HAL. Many people believe this is because if you shift each letter of HAL one more place in the alphabet, you get IBM — the multi-bazillion dollar company.
We would like to know if there are other names with this interesting property. As such, we need you to write a program that accepts a string, shifts each of its characters to the next one in the alphabet, and outputs the resulting string.

Input

The input consists of a single string. You may assume that the string contains neither the character 'Z' nor 'z', nor does it contain any spaces.

Output

Display the string that results when each character in the given string is shifted to its following letter in the alphabet.

Sample Input

```
HellO
```

Sample Output

```
IfmmP
```

Submission 4: Vowel Count

Vowels are a useful feature of the English language[Citation required]. Write a program that counts how many vowels there are in a bunch of words. For our purposes, the letter 'y' is not a vowel. (And, for any Welsh students, neither is 'w'.)

Input

The input consists of lines of text, terminated by a line with the value `end`. Each line may contain both upper- and lower-case letters, **as well as spaces**. Your program should terminate when it accepts `end` as input.

Output

For each line, print out the number of vowels the given line contained.

Example Input-Output

Sample Input

```
Abba Rules!
rhythm
COMS is better than CAM :)
end
```

Sample Output

```
4
0
6
```