## Ex. 1.

Create a program to calculate the final grade of a given student on a course. The final grade is calculated in the following way:

- Test 1: 40%
- Test 2: 20%
- Best mark on 3 labs: 20%
- Second best mark on the 3 labs: 15%
- Lowest mark on 3 labs: 5%

The program should ask the user to give the 5 grades and then print the final course grade in the form:

```
"Final Grade = 14.15"
```

*Hint*: There are two python functions *max* and *min* that return, respectively, the highest and the lowest of a set of parameters:

```
max(150, 30, 43, 80) will return 150
min(150, 30, 43, 80) will return 30
```

## Ex. 2.

Write a program that will read an integer number and print a new integer corresponding to the sum of all digits of the previous one. For example:

```
Give me a number: 9854
The sum of all digits is 26
```

## Ex. 3.

Write a program that will read an integer number and print a new integer corresponding to the inverted digits of the previous one. For example:

```
Give me a number: 98545674
The inverted number is 47654589
```

Or:

```
Give me a number: 4589
The inverted number is 9854
```

## Ex. 4.

Create an empty list. Then ask the user to input a set of integer positive numbers, ending with “q” to stop imputing.

Then ask the user for a new number and test whether that number exists in the list and, if so, how many times. For example, if the user gives the numbers 4, 2, 4, 3, 5, 7 and then the number 4, the program should answer “The number 4 exists in the list 2 times”; if the user gives the number 5, the program should print “The number 5 exists in the list 1 time” (pay attention to plural vs singular forms). If the user enters 8, the program should print “The number 8 does not exist”

## Ex. 5.

Create a Python function that will receive 2 input parameters: a value and a List. The function should return another list containing the positions the value appears in the list or the text “NA” if the value does not appear in the given list.

## Ex. 6.

Create a function called “`HasRepeatedItems(aList)`” that receives a list as a parameter and returns True or False weather the list has repeated values or not.

Hint: There is a Python function that converts a list into a set: `mySet = set(myList)`

## Ex. 7.

Create a Python function called “*CountElements( aList, element )*”. The function must, **without using the Python native function** “*Count()*”, return the number of times “*element*” occurs in the list. Some Examples:

Calling:

```
CountElements( ["a", "b", "a", "c", "a", "b", "c"], "a" )
```

Should return the value: 3

Calling:

```
CountElements( ["a", "b", "a", "c", "a", "b", "c"], "c" )
```

Should return the value: 2

## Ex. 8.

Create a Python function called "*GenerateFibonacci( numElements )*". This function will take a number as input parameter and return the list of the first numbers of the famous Fibonacci sequence. As a reminder, the Fibonacci sequence is a list of numbers in which every element corresponds to the sum of the two previous ones: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, … etc.

Thus, when calling the function:

```
GenerateFibonacci( 12 )
```

Will return the following list:

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

Which corresponds to the first 12 elements of the Fibonacci sequence.

Calling the function:

```
GenerateFibonacci( 20 )
```

Will return the following list:

```
[ 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,
1597, 2584, 4181, 6765]
```

Which corresponds to the first 20 elements of the Fibonacci sequence.

## Ex. 9.

Create a function "`is_prime(n)`" that returns True if the given number "n" is a prime number and False otherwise.

To test your function, write all the prime numbers from 1 to 100.

## Ex. 10.

Create a Python function called "*FactorMyNumber( myNum )*". This function will take a number as an input parameter and must return the list of prime numbers that, when multiplied by each other, will result in the given number. Some examples:

Calling the function:

```
FactorMyNumber( 1925 )
```

Will return the following list:

```
[ 5, 5, 7, 11 ]
```

Because 5 x 5 x 7 x 11 = 1925.

Calling the function:

```
FactorMyNumber( 1820 )
```

Will return the following list:

```
[ 2, 2, 5, 7, 13 ]
```

Because 2 x 2 x 5 x 7 x 13 = 1820.

## Ex. 11.

Create a function called "*SortMyList( aList )*". This function should return a copy of the list passed as paramenter but sorted ascending. Do this **without using the native method** "*sort()*".

**Hint**: Create a new list with the first element of the given list. Then run through the rest of the given list and, for each element, go through the new list and compare it to each of the elements in that new list. If you find one that is greater, insert the new element in the list before that one. Otherwise, append the element to the end of the list.

Useful List Methods:

**`append(e)`** – Adds the element "e" to the end of the list

***insert(i, e)*** – Inserts the element "e" in the list at the position "I"

***pop( )*** – returns the last element of a list and removes it from the list

DO NOT USE THE METHOD ***sort( )***

## Ex. 12.

Consider the function for calculating $e^x$:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Note that each term in the series (that is, for each value of "n") can be calculated from the previous term (term for n-1) , by multiplying it by $\frac{x}{n}$ .

The first term (when n = 0) is 1 : $\frac{x^0}{0!} = 1$

a)

Create a function called "`term_of_series(x, n)`" that returns the value of the n[th] term in the series. Use the recursive form, thus making the function simpler to define**.**

b)

Write a python program that loops in a cycle. Within this cycle, the program must request the value of `x` and the value of `n` and calculate the sum from 0 to n of the terms of the previous series (function `term_of_series(x, n)` defined in the previous task).

The program should then print the following sentence (assuming that the values of "x" and "n" are given):

```
The approximate value of the exponential of x.xx with n elements calculated is yy.yyyy
```

In which x.xx is the value passed as the "x" parameter to the program and "yy.yyyy" the result of calculating the sum of the function terms from 0 to the value "n" (x is displayed with 2 decimal places and the result to 4 decimal places).

The cycle should only end when the user enters an invalid value, for example a letter. In this case, the program prints the following sentence before finishing:

```
You entered an invalid value

     Goodbye...
```

Hint: You can use "`try / except`"

## Ex. 13.

### a)

Write a function named "`leap(y)`" that determines whether a year is a leap year. A year is a leap year if it is divisible by 4 and not divisible by 100 unless it is also divisible by 400. For example, 1984 is leap year, 1100 is not, and 2000 is leap year.

### b)

Write a program that asks the user for a year and answers, using the previous function, whether or not that year is a leap year. This question must be repeated in a cycle until the user types "q" to exit. Any response from the user that is not a number or the letter "q" must print the text "`Answer not accepted. Try again.`"

Hint: you can use "`try / except`"

## Ex. 14.

Implement in Python a function equivalent to the Excel "sumproduct" function. The function receives as parameters 2 lists of values and returns the sum of the products of the values in the equivalent positions. Example:

```
     sumproduct([10, 20, 30, 40, 50], [1, 2, 3, 4, 5])
```

Would return 550 (10*1 + 20*2 + 30*3 + 40*4 + 50*5)

## Ex. 15.

Implement in Python the function equivalent to the Excel function "NPV" (Net Present Value of a set of cash-flows).

Like the Excel function, this npv function should receive as a parameter a list of values where the first value is the interest rate (RATE) and the following correspond to list of yearly cash-flows. Thus, for example, calling:

```
npv([0.055,-58500.00,-19500.00,-20000.00,-10000.00,48000.00,48000.00,55000.00])
```

Should return 11 272,77

## Ex. 16.

Implement in Excel a function that will convert Miles Per Gallon to Liters per 100 Km for automobile fuel consumption.

E.g.:

```
mpg2lp100km(37)
```

Should return 6,357151263647326

Notes:

- 1 mile = 1,609344 Km;
- 1 Gallon (U.S.) = 3,785412