Lab. 3 Functions; WHILE loops

For the exercises below, use the Spyder IDE. Make sure you use a working directory in your computer (preferably that you created in the previous lab) and select it in the File Explorer window of Spyder. In the Editor window create a file "lab3.py" and define the functions with the signatures below. Test the functions created from the console, after importing the file with command "import lab3".

1. Exponential Function

As you know, the exponential function can be computed with the series

$$e(x) = 1 + x + x^2/2! + x^3/3! + x^4/4! + x^5/5! + \ldots$$

Specify function **expo(x)** that implements an approximation of this function and compare it with the predefined function exp/1.

Note: This series converges very quickly (for small values of x) so assess the effect of truncating it with a limited number of terms, either using a fixed number of steps (using a FOR instruction) or a variable number depending on the approximation achieved (i.e. when the first term not considered is less than a certain small value, e.g. 10^{-7}).

2. Logarithm of 2

As you might know, the series below

$$ln(2) = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + \ldots$$

converges (slowly) to **ln(2).** Implement the (constant) function ln2() truncating it in the first term with absolute value less than a certain small value, e.g. 10-7. Since the series is alternate, the approximation error is less than the first neglected term

3. Sine and Cosine

a) Implement function **seno(x)** (x in radians; assume $0 \leq x \leq pi/2$) which approximates the **sin/1** function through the truncated series

$$seno(x) = x - x^3/3! + x^5/5! - x^7/7! + x^9/9! - \ldots$$

b) Do the same for the cosine function approximated by the truncated series

$$coseno(x) = 1 - x^2/2! + x^4/4! - x^6/6! + x^8/8! - \ldots$$

c) Adapt the functions above, to those with the signature below, to so that the input parameter is given in degrees.

```
seno_g(x)
coseno_g(x)
```

d) Optimise the implementation of the functions so that the arguments are converted first in the range 0 .. 2π.

4. Finding values in an array

a) Specify function **find_d(x, V)** that returns the position of the 1st occurrence of value x in vector V. If there is no such position return -1.

b) Generalise the previous function to **find_kd(x, V, k)** that returns the position of the kth occurrence of value x in vector V. If there is no such position return -1.

Examples: Given `V =[ 1,2,4,7,3,9,9,0,1,3,7,1,6]`

```
find_d(7,V) -> 4          find_kd(7,V,1) -> 4
find_d(9,V) -> 6          find_kd(7,V,2) -> 11
find_d(6,V) -> 13         find_kd(7,V,3) -> 0
find_d(8,V) -> 0          find_kd(8,V,1) -> 0
```

- Suggestion: To implement **find_kd(x, V, k)** use an *auxiliay* function **find_kd(x, V, k, z)** that finds the kth occurrence of x in vector V, starting in position z.

c) Adapt the codes above to implement functions **find_r(v, V, k)** and **find_kr(v, V, k)** that returns the indices of the values, but counting backwards.

Examples Given `V =[ 1 2 4 7 3 9 9 0 1 3 7 1 6]`

```
find_r(7,V) -> 11         find_kr(7,V,1) -> 11
find_r(9,V) -> 7          find_kr(7,V,2) -> 4
find_r(6,V) -> 13         find_kr(7,V,3) -> 0
find_r(8,V) -> 0          find_kr(8,V,1) -> 0
```