Lab. 3 Functions; WHILE loops

Do the exercises below in the Octave IDE. You should only use assignments operations with arithmetic expressions excluding pre-defined MATLAB functions. Also use scripts to avoid "too much typing".

1. Exponential Function

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

```
e(x) = 1 + x + x²/2! + x³/3! + x⁴/4! + x⁵/5! + ...
```

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 know, the series below

```
ln(2) = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + ...
```

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 less than the first neglected term

3. Sine and Cosine

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

```
seno(x) = x – x3/3! + x5/5! – x7/7! + x9/9! - ...
```

b) Adapt the function to specify function `seng(x)` that takes the argument in degrees.
c) Do the same for the cosine function approximated by the truncated series

```
coseno(x) = 1– x2/2! + x4/4! – x6/6! + x8/8! - ...
```

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 array V. If there is no such position return 0.
b) Generalise the previous function to `find_kd(x, V, k)` that returns the position of the kth occurrence of value x in array V. If there is no such position return 0.

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
```

c) Adapt the codes 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
```