NUME

***Nume* is module that allows easy calculations on engineering numbers used in electronics and other fields of science and engineering. You can calculate the full range of *yokto* (10e-24) to *jotta* (10e + 24), combined with the standard *math* module we get a powerful python calculation tool. The nume module, which is actually an electronics-focused calculator, can be used in your own scripts or in prototyping in the console, IDLE or other IDE.**

The *nume.py* module is based on the *EEngine.py* module which can also be used in your own scripts. The package uses no additional dependencies other than the python *standard libraries*.

Nume.py and *Eengine.py* are open source modules, if you get measurable financial gains by using *nume*, please support me.
The package in its current state is under constant development. I can add new functions to it, but they won't change the current structure of the nume package, the same should apply to your code.
I am open to new useful functions, please let me know.
Even though the package is in use all the time, some bugs may appear, please let me know.

As the author of the *nume* package, I do not take any responsibility for, that you have suffered a financial loss because of this package, that you missed something, that you cannot know something, that your computer has broken down, for all your failures due to the use of the *nume* package.

REQUIREMENTS

x64 Linux, Windows10+, MacOs,
Python 3+.

INSTALLATION

Linux, MacOs, Windows:
Eengine.py modules, *nume.py* need to be copied to the directory (your own or general) containing the python modules.

The *ss.py* script can be of help to specify the boot directory on Linux systems. If our modules are in a different directory, and preferably in /home/name_user/bin/python, it is worth running the *ss.py* script from there, thanks to which we will get the start path to this directory:

```
> python3 ss.py
```

TUTORIAL - EASY

Before using the nume module in IDLE, the *nume* module needs to be imported:

```
> import nume as e
or
> import nume
or
> import nume*
```

Calculation of current of resistor R1 of 5.1kΩ:

```
> v= 12                    #voltage 12V
> r1= e.k(5.1)             #resistor 5.1kΩ
```

then calkulate:

```
> i= v/r1
> e.unit(i)               #e.unit(v/r1)
(2.352941176470588, 'm')  #current ≈2.35mA
```

Converting the obtained number to the scientific format:

```
> e.sci(i)                #lub e.sci((2.352941176470588, 'm'))
'2.352941176470588e-03'
```

Calculation of resistor limiting the current to 50mA at voltage of 12V:

```
> r= v/e.m(50)            #R= U/50mA
> e.unit(r)
(240.0, '')               #240.0Ω
```

however, we do not have such a resistor, so the closest value in the E12 series will be:

```
> e.valofrow(i)
220.0
```

If this value does not suit us, we need to use a parallel connection, e.g. two resistors. We have a 510Ω resistor and we are looking for a second resistor for 240Ω parallel resistance:

```
> e.parajointfind(240, 510)
453.3333333333333
```

however, we did not find the correct value in E12 series. Only in the series of E96:

```
> e.valofrow(453)         #value is inappropriate
470.0
> e.valofrow(453, 'E96')  #value ok
453.0
```

Calculation of the 22n capacitor impedance at 500Hz and 1.5kHz:

```
> import nume*
> import math
> for freq in (k(0.5), k(1.5)):
      print(unit(1/(2*math.pi*freq*n(22))))

(14.468631190172303, 'k')
(4.822877063390768, 'k')
```

The value of 5.1k, 500k expressed in M():

```
> k(5.1)/M(1)
0.0051                    # 0.0051MΩ
> k(500)/M(1)
0.5                       #0.5MΩ
> 500/m(1)                #500 expressed in m(0.001)
500000.0                  #500000m
```

Generating a list of numbers ranging from smallest to largest:
running a script in the console, or simply clicking on the script icon

```
> python3 Nota-numbers.py
```

FUNCTION description of NUME

Functions units

Functions ***units*** *y* (*yokto*), *z* (*zepto*), *a* (*atto*), *f* (*femto*), *p* (*pico*), *n* (*nano*), *u* (*micro*), *m* (*mili*), *k* (*kilo*), *M* (*mega*), *G* (*giga*), *T* (*tera*), *P* (*peta*), *E* (*eksa*), *Z* (*zetta*), *Y* (*jotta*) used to link values to engineering units.
Argument:

number – number decimal type int, float

```
    k(number)
```

```
> k(1.4)     #1.4kΩ
1400.0
> k(0.00014)
0.13999999999999999
```

unit()

The ***unit()*** function converts decimal number to value with the appropriate unit.
Argument:

number – number decimal type int, float

```
    unit(number)
```

```
> unit(4700)
(4.7, 'k')
> unit(float('4.7e+03'))
(4.7, 'k')
> unit(k(0.0000014))
(1.4, 'm')
> unit(0.000014)
(14.0, 'u')
```

sci()

The ***sci()*** function converts decimal or SI number to the scientific format number.
Argument:

number – number decimal type int, float

```
    sci(number)
```

```
> sci(4700)
'4.7e+03'
> sci((4.7, 'k'))
'4.7e+03'
> sci(k(5.1))
'5.1e+03'
```

valofrow()

The ***valofrow()*** function selects the closest value from the indicated resistance series (E6, E12, E24, E48, E96). If the value exceeds the value contained in the series, the function returns the smallest or

largest value in the series.
Arguments:

value – number decimal type int or float
rowE – series name of type str, where series 'E12' is the default series

```
valofrow(value, rowE='E12')
```

```
> valofrow(4.75)
4.7
> valofrow(4.75e3)
820.0
> valofrow(4.75, 'E48')
4.64
```

parajoint()

The ***parajoint()*** function computes the value of parallel or series connection.
Arguments:

listvalue – list of values for parallel or series connection
conn – type of connection:
- 0 – parallel connection (default)
- 1 – serial connection

```
parajoint(listvalue, conn=0)
```

```
> parajoint([5.1e3, 2.1e3])      lub parajoint([5100, 2100]) #parallel
1487.5
> parajoint([5100, 2100], 1)     #serial
7200
```

parajointfind()

The ***parajointfind()*** function computes the missing value for the resultant parallel connection.
Arguments:

valuepararell – computed value for parallel connection
values – value or list of existing values, decimal value, or list of values

```
parajointfind(valuepararell, values)
```

```
> parajointfind(240, 510)
453.3333333333333
> parajointfind(240, [1200, 1200])
400.00000000000006
```

constans

Constants ***E6***, ***E12***, ***E24***, ***E48***, ***E96*** are lists containing values of resistance series.

```
> E6
(1.0, 1.5, 2.2, 3.3, 4.7, 6.8, 10.0, 15.0, 22.0, 33.0, 47.0, 68.0, 100.0, 150.0, 220.0,
330.0, 470.0, 680.0)
```

list of numbers

The list of numbers was generated by the *Nota-numbers.py* script.

0.000000000000000000000001	1e-24	1y
0.00000000000000000000001	1e-23	10y
0.0000000000000000000001	1e-22	100y
0.000000000000000000001	1e-21	1z
0.00000000000000000001	1e-20	10z
0.0000000000000000001	1e-19	100z
0.000000000000000001	1e-18	1a
0.00000000000000001	1e-17	10a
0.0000000000000001	1e-16	100a
0.000000000000001	1e-15	1f
0.00000000000001	1e-14	10f
0.0000000000001	1e-13	100f
0.000000000001	1e-12	1p
0.00000000001	1e-11	10p
0.0000000001	1e-10	100p
0.000000001	1e-9	1n
0.00000001	1e-8	10n
0.0000001	1e-7	100n
0.000001	1e-6	1u
0.00001	1e-5	10u
0.0001	1e-4	100u
0.001	1e-3	1m
0.001	1e-2	10m
0.01	1e-1	100m
0.0	1e0	1
10	1e+1	10
100	1e+2	100
1000	1e+3	1k
10000	1e+4	10k
100000	1e+5	100k
1000000	1e+6	1M
10000000	1e+7	10M
100000000	1e+8	100M
1000000000	1e+9	1G
10000000000	1e+10	10G
100000000000	1e+11	100G
1000000000000	1e+12	1T
10000000000000	1e+13	10T
100000000000000	1e+14	100T
1000000000000000	1e+15	1P
10000000000000000	1e+16	10P
100000000000000000	1e+17	100P
1000000000000000000	1e+18	1E
10000000000000000000	1e+19	10E
100000000000000000000	1e+20	100E
1000000000000000000000	1e+21	1Z
10000000000000000000000	1e+22	10Z
100000000000000000000000	1e+23	100Z
1000000000000000000000000	1e+24	1Y