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:
The *eengine.py*, *nume.py* modules need to be copied to the modules directory shared by python.

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
```

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.

Arguments:
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.

Arguments:
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(0.00001
(14.0, 'u')
```

sci()

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

Arguments:
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. The series argument can be given with the lower case letter 'e24'.

```
.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
```

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)
```