Functionality

The core features of the package are shown in the following paragraphs.

```python
import engicalc as e
import numpy as np
```

Integration of Pint units

Common Units

Common units are stored as variables and are automatically imported with the package. The following units are available:

```python
for unit in e.units.items():
    print(unit[1])
```

```
kg
t
kNm
Nm
N
kN
MN
m
cm
dm
mm
km
rad
deg
%
‰
s
°C
K
MPa
```

Handling Units

Units can be added using operators:

```python
v = 30 *e.kNm
```

```
v_t = v + 30*e.kNm
```

You can also access the full unit registry:

```
u = 400 *e.ureg.hours

print(u)
```

```
400 h
```

Convert units using Pint:

```
u.to(e.s)
```

1440000 s

Parsing

Cell Parsing

A simple parsing function extracts calculations from a cell:

```
v = 30 *e.kNm

v_t = v + 30*e.kNm

e.parse_cell()
```

```
[{'variable_name': 'v',
  'expression': '30 * e.kNm',
  'result': <Quantity(30, 'kNm')>},
 {'variable_name': 'v_t',
  'expression': 'v + 30 * e.kNm',
  'result': <Quantity(60, 'kNm')>}]
```

Drawbacks

- The parsing function loses datatype information, returning rows as strings.
- Currently only assignments work, no conditionals, or other python syntax

```
b = 20

if b >20:
    b = 40
```

```
e.parse_cell()
```

```
[{'variable_name': 'b', 'expression': '20', 'result': 20},
 {'variable_name': 'b', 'expression': '40', 'result': 40}]
```

Markdown rendering

Parsed cell content is processed using `sympy.sympify` and converted to `latex` via the `sympy` latexprinter. Eventually the `latex`code is inserted into a Markdown math environment. The `render()`function is capable of the following:

Numeric representation

The numeric representation shows the Variablename and its value.

```
v = 30 *e.kNm

v_t = v + 30*e.kNm


Theta_pl_A= 5
m_u_A = 10
m_y_A=5
l=1
b_w=0.1
EI_II=20
q_S1_A= 10

e.render(symbolic=True)
```

$$\begin{aligned} v &= 30 \cdot \mathrm{kNm} = 30\,\mathrm{kNm} \\ v_t &= v + 30 \cdot \mathrm{kNm} = 60\,\mathrm{kNm} \\ \Theta_{pl A} &= 5 \\ m_{u A} &= 10 \\ m_{y A} &= 5 \\ l &= 1 \\ b_w &= 0.1 \\ EI_{II} &= 20 \\ q_{S1 A} &= 10 \end{aligned}$$

Symbolic Representation

The symbolic representation is also showing the calculation.

```
alpha_u_A = np.sqrt(Theta_pl_A) / 2
Delta = ((np.sin(alpha_u_A) + (m_u_A - m_y_A) * l * b_w / (3 * EI_II)) * 24 *
EI_II / l**3)*e.m
q_u_A = Delta + q_S1_A*e.m
e.render(raw=False)
```

$$\alpha_{uA} = \frac{\sqrt{\Theta_{plA}}}{2} = 1.12$$

$$\Delta = \left(\sin(\alpha_{uA}) + \frac{(m_{uA} - m_{yA}) \cdot l \cdot b_w}{3 \cdot EI_{II}}\right) \cdot 24 \cdot EI_{II} \cdot \frac{1}{l^3} \cdot \mathrm{m} = 435.64\,\mathrm{m}$$

$$q_{uA} = \Delta + q_{S1A} \cdot \mathrm{m} = 445.64\,\mathrm{m}$$

Only symbolic

As a sidefunctionality, only the symbolic part can be shown.

```
Delta
l
x = 2*Delta
e.render(symbolic=True, numeric=False)
```

$$\Delta = \left(\sin(\alpha_{uA}) + \frac{(m_{uA} - m_{yA}) \cdot l \cdot b_w}{3 \cdot EI_{II}}\right) \cdot 24 \cdot EI_{II} \cdot \frac{1}{l^3} \cdot \mathrm{m}$$

$$l = 1$$

$$x = 2 \cdot \Delta$$

Recalling variables

The defined variables in the notebook are stored in a container. They can be recalled at anytime and ec.rendered again.

```
v
Delta

e.render()
```

$$v = 30 \cdot \mathrm{kNm} = 30\,\mathrm{kNm}$$

$$\Delta = \left(\sin(\alpha_{uA}) + \frac{(m_{uA} - m_{yA}) \cdot l \cdot b_w}{3 \cdot EI_{II}}\right) \cdot 24 \cdot EI_{II} \cdot \frac{1}{l^3} \cdot \mathrm{m} = 435.64\,\mathrm{m}$$

Multiple Rows

The markdown math environment is capable of displaying the equations in multiple rows:

```
v
Delta
Theta_pl_A
EI_II
v_t
e.render(rows=3, symbolic=False)

e.render(rows=5, symbolic=False)
```

$$v = 30\,\mathrm{kNm} \quad \Delta = 435.64\,\mathrm{m} \quad \Theta_{plA} = 5$$
$$EI_{II} = 20 \quad v_t = 60\,\mathrm{kNm}$$

$$v = 30\,\mathrm{kNm} \quad \Delta = 435.64\,\mathrm{m} \quad \Theta_{plA} = 5 \quad EI_{II} = 20 \quad v_t = 60\,\mathrm{kNm}$$

Numpy functions

The package is based around the numpy functions and they should be used. Currently the `numpy.` is stripped and the numpyfunction gets translated to `sympy` via `sympy.sympify`.

```
alpha = 45*e.deg
test = np.atan(alpha.to(e.los) + 25)

e.render(raw=False)
```

$$\alpha = 45 \cdot {}^{\circ} = 45\,{}^{\circ}$$
$$test = \operatorname{atan}\left(\alpha + 25\right) = 1.53\,\mathrm{rad}$$

An array is translated to a matrix.

```
F_x = np.abs(np.array([-13,30,23,12])*e.kN)
F_v = F_x * 2
F_z = F_x*np.atan(alpha)**np.sqrt(1)
F_y = F_x * F_v
e.render(symbolic=True)
```

$$F_x = \left| \begin{bmatrix} -13 \cdot \mathrm{kN} \\ 30 \cdot \mathrm{kN} \\ 23 \cdot \mathrm{kN} \\ 12 \cdot \mathrm{kN} \end{bmatrix} \right| = \begin{bmatrix} 13 \\ 30 \\ 23 \\ 12 \end{bmatrix} \mathrm{kN}$$

$$F_v = F_x \cdot 2 = \begin{bmatrix} 26 \\ 60 \\ 46 \\ 24 \end{bmatrix} \mathrm{kN}$$

$$F_z = F_x \cdot \mathrm{atan}^{\sqrt{1}}(\alpha) = \begin{bmatrix} 8.66 \\ 19.97 \\ 15.31 \\ 7.99 \end{bmatrix} \mathrm{kN} \cdot \mathrm{rad}$$

$$F_y = F_x \cdot F_v = \begin{bmatrix} 338 \\ 1800 \\ 1058 \\ 288 \end{bmatrix} \mathrm{kN}^2$$

Raw Markdown

As the markdowncode is stored anyways, it can be output aswell. Could be used to copy into a table.

```
v
Delta
Theta_pl_A
e.render(symbolic=True, raw=True)
```

```
'$$\\begin{aligned}v& = 30 \\cdot \\mathrm{kNm} = 30 \\ \\mathrm{kNm} \\\\ \
\Delta& = \\left(\\sin{\\left(\\alpha_{u A} \\right)} + \\frac{\\left(m_{u A}
- m_{y A}\\right) \\cdot l \\cdot b_{w}}{3 \\cdot EI_{II}}\\right) \\cdot 24 \
\cdot EI_{II} \\cdot \\frac{1}{l^{3}} \\cdot \\mathrm{m} = 435.64 \\ \
\mathrm{m} \\\\ \\Theta_{pl A}& = 5\\end{aligned}$$'
```

Special Characters

Some special characters are inserted in the string before the sympy conversion takes place. In the `output` module, a replacement dictionary is created to replace the special characters. This can be expanded. It has to correspond to the `latex` syntax.

```
diam_infty = 20

infty__infty_infty__diam = 10
e.render(raw=False)
```

$$
\begin{aligned}
\oslash_{\infty} &= 20 \\
\infty_{\infty}^{\infty^{\oslash}} &= 10
\end{aligned}
$$

Pint unit handling

```
v = v_t.to(e.Nm) + 30*2*e.Nm
q = v_t.magnitude*e.Nm + v
q__shortcut = v_t.m * e.Nm + v
e.render()
```

$$
\begin{aligned}
v &= v_t + 30 \cdot 2 \cdot \mathrm{Nm} = 60060.0\,\mathrm{Nm} \\
q &= v_t \cdot \mathrm{Nm} + v = 60120.0\,\mathrm{Nm} \\
q^{shortcut} &= v_t \cdot \mathrm{Nm} + v = 60120.0\,\mathrm{Nm}
\end{aligned}
$$

Functions

It can be useful to display the calculations that have been done in a function environment. For that there is a second parsing function. The Funcion parses the local variables.

```
from IPython.display import Markdown
```

```
def test(alpha__top, b):
    x = alpha__top + b
    y = alpha__top-b*2
    z = x + y
    display(Markdown('**Only Symbolic representation**'))
    e.render_func(numeric=False ,rows=2)
    display(Markdown('**Whole representation**'))
    e.render_func(numeric=True ,rows=3)
    return z
```

```
z = test(3,4)
```

Only Symbolic representation

$$
\begin{aligned}
\alpha^{top} &= 3 & b &= 4 \\
x &= \alpha^{top} + b & y &= \alpha^{top} - b \cdot 2 \\
z &= x + y
\end{aligned}
$$

Whole representation

$$
\begin{aligned}
\alpha^{top} &= 3 & b &= 4 & x &= \alpha^{top} + b = 7 \\
y &= \alpha^{top} - b \cdot 2 = -5 & z &= x + y = 2
\end{aligned}
$$

```
z__2 = test(5*e.kNm + 3*e.kNm, 3*e.kNm)
```

Only Symbolic representation

$$
\begin{aligned}
\alpha^{top} &= 8\,\mathrm{kNm} & b &= 3\,\mathrm{kNm} \\
x &= \alpha^{top} + b & y &= \alpha^{top} - b \cdot 2 \\
z &= x + y
\end{aligned}
$$

Whole representation

$$
\begin{aligned}
\alpha^{top} &= 8\,\mathrm{kNm} & b &= 3\,\mathrm{kNm} & x &= \alpha^{top} + b = 11\,\mathrm{kNm} \\
y &= \alpha^{top} - b \cdot 2 = 2\,\mathrm{kNm} & z &= x + y = 13\,\mathrm{kNm}
\end{aligned}
$$

Drawback

The `render` function parses the assignments inside the function.

```
def test2(a, b):
    x = a + b
    y = a**b
    z = x + y
    return z

e.render()
```

$$
\begin{aligned}
x &= a + b = None \\
y &= a^{b} = None \\
z &= x + y = None
\end{aligned}
$$

```
z = test2(3,4)

e.render()
```

$$z = \mathrm{test}_{2}(3, 4) = 88$$

Markdown tables

It can be useful to summarize the calculations in a table. For that the variables can easily be inserted into the table. Using the function `render_list`.

```
import pandas as pd

# Example lists
col_1 = e.render_list([z, v, q__shortcut, v_t], numeric=False, raw=True)
col_2 = e.render_list([z, v, q__shortcut, v_t], symbolic=False, raw=True)
names = ['Höhe', 'Differenz', 'Test', 'Test3']
```

```
# Define column names
columnnames = ['Bezeichnung', 'Berechnung', 'Berechnung2']

# Create DataFrame
DF = pd.DataFrame(list(zip(names, col_1, col_2)), columns=columnnames)

# Display the DataFrame
display(Markdown(DF.to_markdown(tablefmt='pipe', index=False)))
```

Beze-ich-nung	Berechnung	Berechnung2
Höhe	$z = \mathrm{test}_2(3, 4)$	$z = 88$
Differenz	$v = v_t + 30 \cdot 2 \cdot \mathrm{Nm}$	$v = 60060.0\ \mathrm{Nm}$
Test	$q^{shortcut} = v_t \cdot \mathrm{Nm} + v$	$q^{shortcut} = 60120.0\ \mathrm{Nm}$
Test3	$v_t = v + 30 \cdot \mathrm{kNm}$	$v_t = 60\ \mathrm{kNm}$