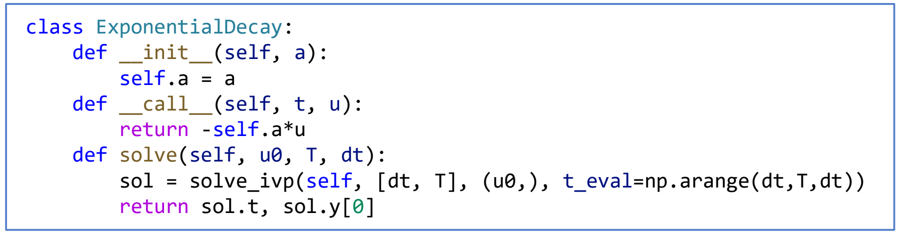
Oppgave 1

- Problemstillingen
- Part 1 - Solving Initial Value Problems
    - Clas for Ode, solving ode, testing.
    - $\frac{du}{dt} = -au$
- ODE SolverPart 2 — The Single Pendulum
    - Class for the system, solving equations, properties, cartesian, Energi conservation, Dampened Pendulum.
    - $\frac{d\theta}{dt} = \omega$, $\frac{d\omega}{dt} = -\frac{g}{L}\sin(\theta)$ og $\frac{d\omega}{dt} = -\frac{g}{L}\sin(\theta) - \frac{B}{M}\omega$
- Part 3 — The Double Pendulum
    - Solving the equations, properties, Testing
- Part 4 — Animating the pendulum
    - _next_frame, Interface, creating the animation.

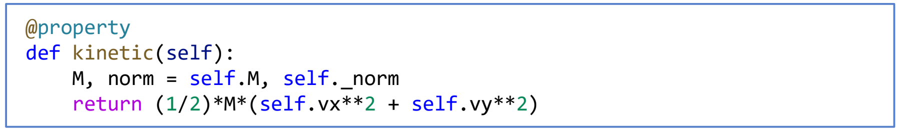
- Temaer
- ODE
- Properties
- Private verdier
- Vektorisering
- Dokumentasjons lesing
- Unittest
- Plot og animasjon
- Arv

Svar på problemstillingen

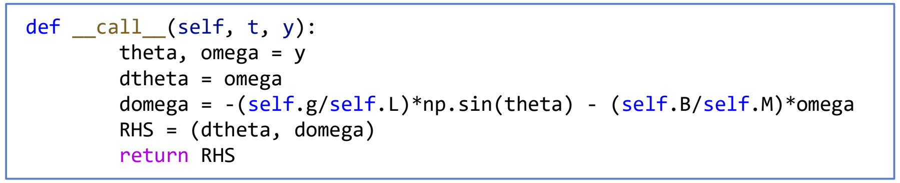
- Problemstillingen
- Part 1 - Solving Initial Value Problems
  - Test: tester ved å bruke abs(expected – calculated) < eps. Testet også for falsk positiv.
- Part 2 — The Single Pendulum
  - Veldig lik part 1, tester for forventet verdi og for stasjonert punkt. Bevaring av energi.
  - Ikke perfekt pendel
- Part 3 — The Double Pendulum
  - Lik part 2, men med større formler. Solve er den samme som 1.
  - Tester: y0=0, not_implemented, null mass.
- Part 4 — Animating the pendulum
  - Create animation: konfigurerer, _next_frame returnerer bilde 1 i riktig format, save animation: larger animasjonen, show animation: viser animasjonen.



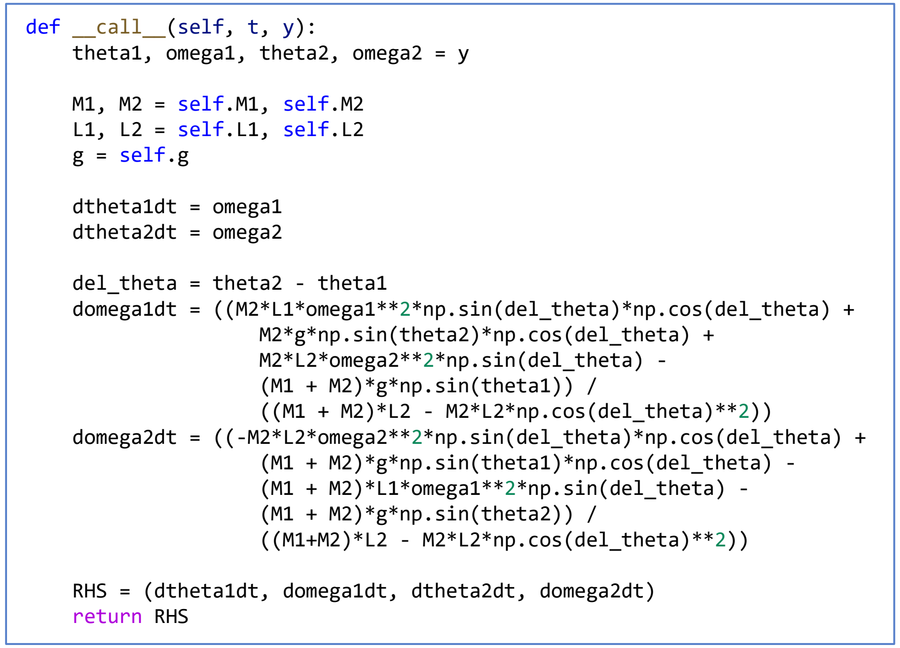
```python
class ExponentialDecay:
    def __init__(self, a):
        self.a = a
    def __call__(self, t, u):
        return -self.a*u
    def solve(self, u0, T, dt):
        sol = solve_ivp(self, [dt, T], (u0,), t_eval=np.arange(dt,T,dt))
        return sol.t, sol.y[0]
```



```python
@property
def kinetic(self):
    M, norm = self.M, self._norm
    return (1/2)*M*(self.vx**2 + self.vy**2)
```



```python
def __call__(self, t, y):
        theta, omega = y
        dtheta = omega
        domega = -(self.g/self.L)*np.sin(theta) - (self.B/self.M)*omega
        RHS = (dtheta, domega)
        return RHS
```

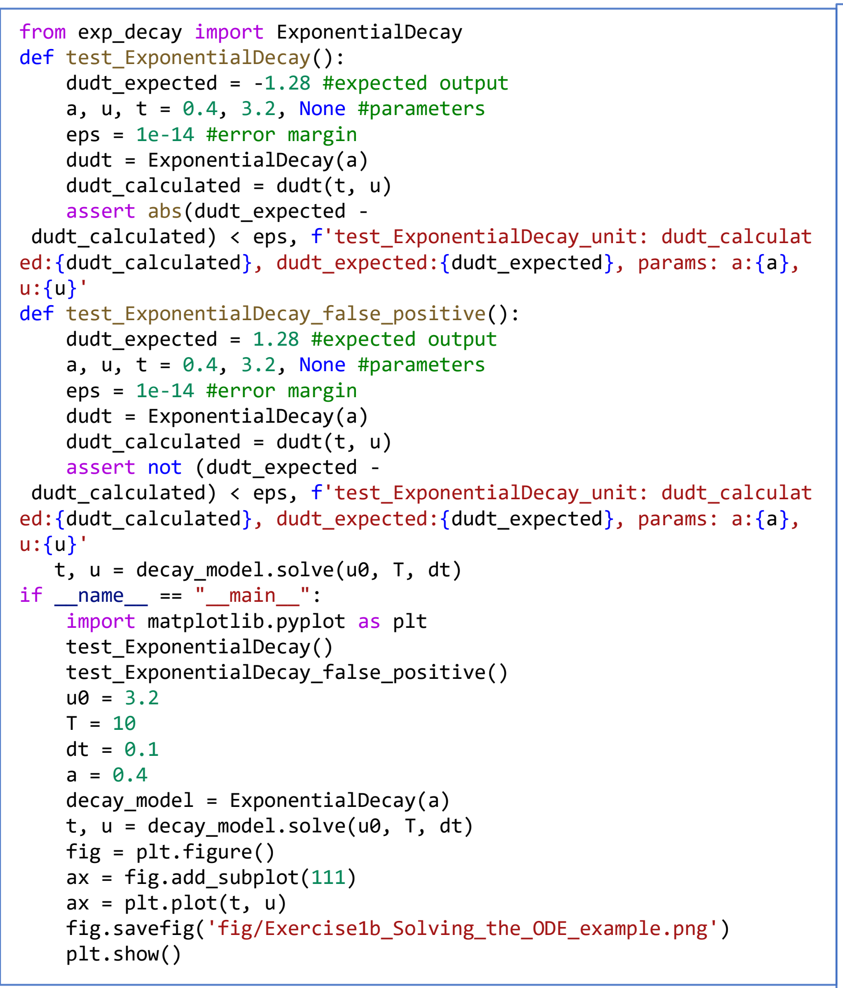


```python
def __call__(self, t, y):
    theta1, omega1, theta2, omega2 = y

    M1, M2 = self.M1, self.M2
    L1, L2 = self.L1, self.L2
    g = self.g

    dtheta1dt = omega1
    dtheta2dt = omega2

    del_theta = theta2 - theta1
    domega1dt = ((M2*L1*omega1**2*np.sin(del_theta)*np.cos(del_theta) +
                    M2*g*np.sin(theta2)*np.cos(del_theta) +
                    M2*L2*omega2**2*np.sin(del_theta) -
                    (M1 + M2)*g*np.sin(theta1)) /
                    ((M1 + M2)*L2 - M2*L2*np.cos(del_theta)**2))
    domega2dt = ((-M2*L2*omega2**2*np.sin(del_theta)*np.cos(del_theta) +
                    (M1 + M2)*g*np.sin(theta1)*np.cos(del_theta) -
                    (M1 + M2)*L1*omega1**2*np.sin(del_theta) -
                    (M1 + M2)*g*np.sin(theta2)) /
                    ((M1+M2)*L2 - M2*L2*np.cos(del_theta)**2))

    RHS = (dtheta1dt, domega1dt, dtheta2dt, domega2dt)
    return RHS
```

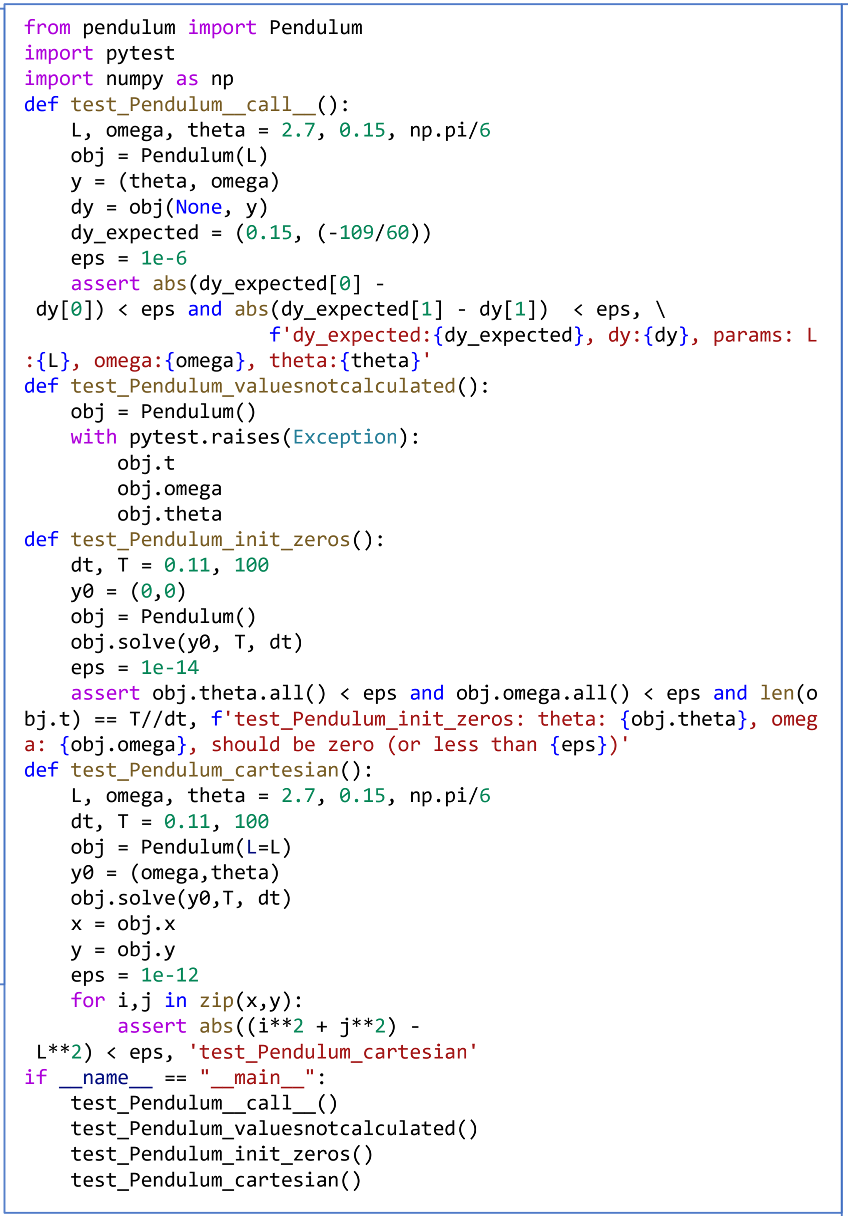
Utfordringer

- Lagring av animasjoner.
  - Måtte installere grafikk motoren selv.
- Git
  - Problemer rundt det å ha 2 github profiler.

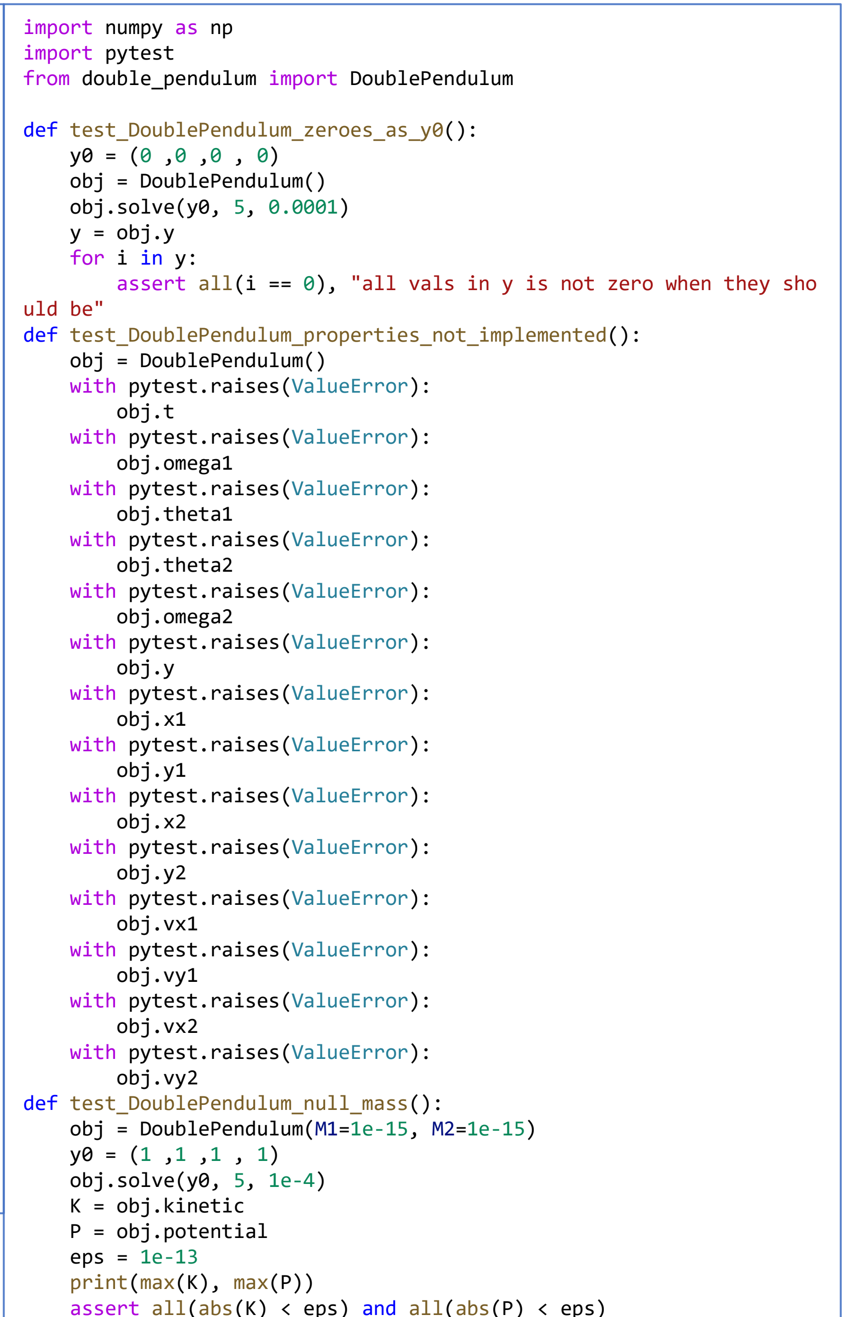
Testing, og sikring at programmet virker



```
from exp_decay import ExponentialDecay
def test_ExponentialDecay():
    dudt_expected = -1.28 #expected output
    a, u, t = 0.4, 3.2, None #parameters
    eps = 1e-14 #error margin
    dudt = ExponentialDecay(a)
    dudt_calculated = dudt(t, u)
    assert abs(dudt_expected -
 dudt_calculated) < eps, f'test_ExponentialDecay_unit: dudt_calculat
ed:{dudt_calculated}, dudt_expected:{dudt_expected}, params: a:{a},
u:{u}'
def test_ExponentialDecay_false_positive():
    dudt_expected = 1.28 #expected output
    a, u, t = 0.4, 3.2, None #parameters
    eps = 1e-14 #error margin
    dudt = ExponentialDecay(a)
    dudt_calculated = dudt(t, u)
    assert not (dudt_expected -
 dudt_calculated) < eps, f'test_ExponentialDecay_unit: dudt_calculat
ed:{dudt_calculated}, dudt_expected:{dudt_expected}, params: a:{a},
u:{u}'
   t, u = decay_model.solve(u0, T, dt)
if __name__ == "__main__":
    import matplotlib.pyplot as plt
    test_ExponentialDecay()
    test_ExponentialDecay_false_positive()
    u0 = 3.2
    T = 10
    dt = 0.1
    a = 0.4
    decay_model = ExponentialDecay(a)
    t, u = decay_model.solve(u0, T, dt)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax = plt.plot(t, u)
    fig.savefig('fig/Exercise1b_Solving_the_ODE_example.png')
    plt.show()
```



```
from pendulum import Pendulum
import pytest
import numpy as np
def test_Pendulum__call__():
    L, omega, theta = 2.7, 0.15, np.pi/6
    obj = Pendulum(L)
    y = (theta, omega)
    dy = obj(None, y)
    dy_expected = (0.15, (-109/60))
    eps = 1e-6
    assert abs(dy_expected[0] -
 dy[0]) < eps and abs(dy_expected[1] - dy[1])  < eps, \
                     f'dy_expected:{dy_expected}, dy:{dy}, params: L
:{L}, omega:{omega}, theta:{theta}'
def test_Pendulum_valuesnotcalculated():
    obj = Pendulum()
    with pytest.raises(Exception):
        obj.t
        obj.omega
        obj.theta
def test_Pendulum_init_zeros():
    dt, T = 0.11, 100
    y0 = (0,0)
    obj = Pendulum()
    obj.solve(y0, T, dt)
    eps = 1e-14
    assert obj.theta.all() < eps and obj.omega.all() < eps and len(o
bj.t) == T//dt, f'test_Pendulum_init_zeros: theta: {obj.theta}, omeg
a: {obj.omega}, should be zero (or less than {eps})'
def test_Pendulum_cartesian():
    L, omega, theta = 2.7, 0.15, np.pi/6
    dt, T = 0.11, 100
    obj = Pendulum(L=L)
    y0 = (omega,theta)
    obj.solve(y0,T, dt)
    x = obj.x
    y = obj.y
    eps = 1e-12
    for i,j in zip(x,y):
        assert abs((i**2 + j**2) -
 L**2) < eps, 'test_Pendulum_cartesian'
if __name__ == "__main__":
    test_Pendulum__call__()
    test_Pendulum_valuesnotcalculated()
    test_Pendulum_init_zeros()
    test_Pendulum_cartesian()
```



```
import numpy as np
import pytest
from double_pendulum import DoublePendulum

def test_DoublePendulum_zeroes_as_y0():
    y0 = (0 ,0 ,0 , 0)
    obj = DoublePendulum()
    obj.solve(y0, 5, 0.0001)
    y = obj.y
    for i in y:
        assert all(i == 0), "all vals in y is not zero when they sho
uld be"
def test_DoublePendulum_properties_not_implemented():
    obj = DoublePendulum()
    with pytest.raises(ValueError):
        obj.t
    with pytest.raises(ValueError):
        obj.omega1
    with pytest.raises(ValueError):
        obj.theta1
    with pytest.raises(ValueError):
        obj.theta2
    with pytest.raises(ValueError):
        obj.omega2
    with pytest.raises(ValueError):
        obj.y
    with pytest.raises(ValueError):
        obj.x1
    with pytest.raises(ValueError):
        obj.y1
    with pytest.raises(ValueError):
        obj.x2
    with pytest.raises(ValueError):
        obj.y2
    with pytest.raises(ValueError):
        obj.vx1
    with pytest.raises(ValueError):
        obj.vy1
    with pytest.raises(ValueError):
        obj.vx2
    with pytest.raises(ValueError):
        obj.vy2
def test_DoublePendulum_null_mass():
    obj = DoublePendulum(M1=1e-15, M2=1e-15)
    y0 = (1 ,1 ,1 , 1)
    obj.solve(y0, 5, 1e-4)
    K = obj.kinetic
    P = obj.potential
    eps = 1e-13
    print(max(K), max(P))
    assert all(abs(K) < eps) and all(abs(P) < eps)
```