```python
# convert_rv_kep
# translate between the state vector and classical keplerian orbit elements
# Inputs:
#   input_flag - type of state being inputted
#   x - the numbers
#   delta_t - time elapsed since t_0
#   output_flag - type of state being outputted


# imports
import numpy as np


# # #
def convert_rv_kep(input_flag, x_vec, delta_t, output_flag):

    # define some constants
    mu_E = 398600.0  # [km^3/s^2] standard gravitational parameter of the Earth

    # Handle various cases, if none then it just returns
    # keplerian to state vector
    if input_flag == 'keplerian' and output_flag == 'state':
        # parse
        a = x_vec[0]  # [km]
        ecc = x_vec[1]  # [none]
        inc = np.deg2rad(x_vec[2])  # [deg]
        Omega = np.deg2rad(x_vec[3])  # [deg]
        omega = np.deg2rad(x_vec[4])  # [deg]
        M_0 = np.deg2rad(x_vec[5])  # [deg]

        if ecc < 1.0:
            M = M_0 + np.sqrt(mu_E / np.power(a, 3.0)) * delta_t
            f = convert_M_to_f(M, 6, ecc)
        else:
            n = np.sqrt(mu_E / np.power(-a, 3.0))
            N = M_0 + n*delta_t
            f = convert_M_to_f(N, 6, ecc)

        theta = omega + f

        p = a * (1.0 - np.power(ecc, 2.0))

        h = np.sqrt(mu_E * p)

        r = p / (1.0 + ecc*np.cos(f))
```

```python

        r_x = r * (np.cos(Omega)*np.cos(theta) - np.sin(Omega)*np.sin(theta)*np.cos(inc))
        r_y = r * (np.sin(Omega)*np.cos(theta) + np.cos(Omega)*np.sin(theta)*np.cos(inc))
        r_z = r * (np.sin(theta)*np.sin(inc))

        v_x = -mu_E / h * (np.cos(Omega)*(np.sin(theta) + ecc*np.sin(omega)) + np.sin(Omega)*(np.cos(theta) + ecc*np.cos(omega))*np.cos(inc))
        v_y = -mu_E / h * (np.sin(Omega)*(np.sin(theta) + ecc*np.sin(omega)) - np.cos(Omega)*(np.cos(theta) + ecc*np.cos(omega))*np.cos(inc))
        v_z = mu_E / h * (np.cos(theta) + ecc*np.cos(omega))*np.sin(inc)

        x_out = np.array([[r_x], [r_y], [r_z], [v_x], [v_y], [v_z]])

        return x_out

    # state vector to keplerian
    elif input_flag == 'state' and output_flag == 'keplerian':
        # parse
        x = x_vec[0]
        y = x_vec[1]
        z = x_vec[2]
        xd = x_vec[3]
        yd = x_vec[4]
        zd = x_vec[5]

        r_vec = np.array([x, y, z])
        v_vec = np.array([xd, yd, zd])

        r = np.linalg.norm(r_vec)
        v = np.linalg.norm(v_vec)

        one_over_a = 2.0 / r - np.power(v, 2.0) / mu_E
        a = 1.0 / one_over_a

        h_vec = np.cross(r_vec, v_vec)
        h = np.linalg.norm(h_vec)

        ecc_vec = np.cross(v_vec, h_vec) / mu_E - r_vec / r
        ecc = np.linalg.norm(ecc_vec)

        ihat_e = ecc_vec / ecc
        ihat_h = h_vec / h
        ihat_p = np.cross(ihat_h, ihat_e)
```

```

        PN = np.array([ihat_e.T, ihat_p.T, ihat_h.T])

        Omega = np.arctan2(PN[2, 0], -PN[2, 1])
        inc = np.arccos(PN[2, 2])
        omega = np.arctan2(PN[0, 2], PN[1, 2])
        ihat_r = r_vec / r
        f = np.arctan2(np.dot(np.cross(ihat_e, ihat_r), ihat_h), np.dot(ihat_e, ihat_r))
        if ecc < 1.0:
            E = 2.0*np.arctan(np.tan(f/2.0) / np.sqrt((1.0 + ecc)/(1.0 - ecc)))
            M = E - ecc*np.sin(E)
            n = np.sqrt(mu_E / np.power(a, 3.0))
        else:
            H = 2.0*np.arctanh(np.tan(f/2) / np.sqrt((ecc + 1.0) / (ecc - 1.0)))
            M = ecc*np.sinh(H) - H
            n = np.sqrt(mu_E / np.power(-a, 3.0))

        M_0 = M - n * delta_t
        if M_0 < 0:
            M_0 = M_0 + 2 * np.pi

        return np.array([[a], [ecc], [np.rad2deg(inc)], [np.rad2deg(Omega)], [np.rad2deg(omega)], [np.rad2deg(M_0)]])


    # flags are the same
    elif input_flag == output_flag:
        return x
    # inputs are wrong
    else:
        raise ValueError('Incorrect input or output flags')


# subroutine for newton's method to solve keplers equation for E (eccentric anomaly)
# Inputs:
#   x_0 - [deg] initial guess
#   n_iter - [none] number of iterations to be completed
#   ecc - eccentricity of orbit
# Outputs:
#   x_k - final solution
def convert_M_to_f(x_0, n_iter, ecc):

    # iterate
    x_k = x_0
```

```python
    for k in range(n_iter):
        # elliptic case
        if ecc < 1.0:
            x_k = x_k - (x_0 - (x_k - ecc*np.sin(x_k)))/-(1.0 - ecc*np.cos(x_k))
        # hyperbolic case
        else:
            x_k = x_k - (x_0 - (ecc*np.sinh(x_k) - x_k))/-(ecc*np.cosh(x_k) - 1)

    # elliptic case
    if ecc < 1.0:
        f = 2.0 * np.arctan(np.sqrt((ecc + 1.0)/(1.0 - ecc)) * np.tan(x_k / 2.0))
    # hyperbolic case
    else:
        f = 2.0 * np.arctan(np.sqrt((ecc + 1.0) / (ecc - 1.0)) * np.tanh(x_k / 2.0))

    return f

```