```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Sep 18 15:59:25 2020

@author: flo
"""

#TO DO: check names of objects, functions and variables to make it more readable


from pylab import *
import numpy as np

# reads out the motor activity of all motors


def read_out_motors_activity(list_of_motors):
    data = []
    for m in list_of_motors:
        data.append(m.attachment_state)
    return data

# reads out the position of all motors


def read_out_motors_position(list_of_motors):
    data = []
    for m in list_of_motors:
        data.append(m.position)
    return data


# Class that defines the binding event
class Binding:
    def __init__(self, rate, motor):
        self.rate = rate
        self.motor = motor

    # Executing an binding event
    def execute(self):
        self.motor.attachment_state = 1

    # Returns the binding rate: if bound, the motor cannot bind anymore,
    # therefore the rate = 0
    def get_rate(self):
        # if bound
        if self.motor.attachment_state == 1:
            return 0
        else:
            return self.rate


# Class that defines the unbinding event
class Unbinding:
    def __init__(self, rate, motor):
        self.rate = rate
        self.motor = motor
```

```
# Executes an binding event
    def execute(self):
        self.motor.attachment_state = 0
        # If motor unbinds, its position is set to zero
        self.motor.position = 0

# Returns the unbinding rate: if bound, return unbinding rate, if unbound, return 0
    def get_rate(self):
        # if bound
        if self.motor.attachment_state == 1:
            return self.rate
        else:
            return 0

# Class that defines the stepping event
class Stepping:
    def __init__(self, rate, motor):
        self.rate = rate
        self.motor = motor

# a stepping event changes the position of the motor by 8 nm
    def execute(self):
        self.motor.position += 8

# returns the stepping rate. Only if the motor is bound it can step
    def get_rate(self):
        # if bound
        if self.motor.attachment_state == 1:
            return self.rate
        else:
            return 0


# Class that defines the motor
class Motor:
    def __init__(self, attachment_state, position):
        self.attachment_state = attachment_state
        self.position = position

# List of all motors
motors = []

# Number of motors in the simulation
num_of_motors = 3

# Initialize all motors into the list
for i in arange(0, num_of_motors):
    # All motors are bound, initializing with attachment_state = 1
    motors.append(Motor(1, 0))

# List of all events
events = []

# For each motor in the list, an binding and an unbinding event is initialzied in the list
for m in motors:
    events.append(Binding(5, m))
    events.append(Unbinding(1, m))
    events.append(Stepping(50, m))
```

```
# simulation time
time = 10
t = 0
data = []

while t <= time:

# sum of all rates
    sum_rates = 0
    for e in events:
        sum_rates += e.get_rate()

# waiting time from exp dist
    dt = np.random.exponential(1/sum_rates)

# Checking which transition should happen
    r = np.random.uniform()

    prob = 0
    for e in events:
        prob += e.get_rate()/sum_rates
        if prob >= r:
            e.execute()
            break

    t += dt
    # md = read_out_motors(motors)

    pos = read_out_motors_position(motors)
    data.append([t] + pos)

data = asarray(data)

# Plotting
plt.figure()
for i in arange(1, len(data[0])):
    plt.plot(data[:, 0], data[:, i], label='motor ' + str(i))

plt.legend()
plt.xlabel('time (s)')
plt.ylabel('position (nm)')
plt.show()

# A figure is saved in /results/figures/
savefig('../results/figures/R2_k11_10-5.pdf', bbox_inches='tight')
```