```python
"""
Created on Thu Jan 25 18:30:59 2018

@author: Jason Grillo, Thomas Goehring, Trent Peterson
"""

class Controller:
    ''' This class implements a generic proportional gain controller'''

    def __init__(self, Kp, Ki, Kd, saturation):
        '''
        Initializes the proportional gain and defines the
        initial setpoint for the controller.
        @param Kp: Sets proportional gain value
        @param Ki: Sets integral gain value
        @param saturation: The anti-wind up integration saturation limit
        '''
        print('Creating a controller')
        ## Proportional gain for a control loop
        self.Kp = Kp
        ## Integral gain for a control loop
        self.Ki = Ki
        ## Derivative gain for a control loop
        self.Kd = Kd
        ## Desired output target variable
        self.setpoint = 0
        self.saturation = saturation
        ## Actuation signal sent to the plant
        self.__actuate_signal = 0
        ## Current output value of feedback from plant
        self.__current_value = 0
        ## Set the start variable to true to begin integral gain term
        self.__start = True
        self._esum = 0
        self._perror = 0
        self._deriv = 0

        print('Controller sucessfully created')

    def set_setpoint(self, new_setpoint):
        '''
        Method to enable the user to define a new setpoint that the
        control loop will use as a reference value.
        @param new_setpoint: User-defined setpoint that the controller us
        '''
        self.setpoint = new_setpoint
```

```python
    def set_Kp(self, new_Kp):
        '''
        Method to enable the user to define a new proportional gain
        that the control loop will use to multiply the error signal
        and output an actuation signal.
        @param new_Kp: User-defined propotional gain that is multiplied b
        '''
        self.Kp = new_Kp

    def set_Ki(self, new_Ki):
        '''
        Method to enable the user to define a new integral gain
        that the control loop will use to multiply the error sum signal
        and output an actuation signal.
        @param new_Ki: User-defined integral gain that is multiplied by e
        '''
        self.Ki = new_Ki

    def set_newSat(self, new_Sat):
        '''
        Method to enable the user to define a integral saturation
        and output a saturated error signal if saturated
        @param newSat: User-defined saturation that is used to prevent in
        '''
        self.saturation = new_Sat

    def set_Kd(self, new_Kd):
        '''
        Method to enable the user to define a new derivative gain
        that the control loop will use to multiply the error signal
        and output an actuation signal.
        @param new_Kd: User-defined derivative gain that is multiplied by
        '''
        self.Kd = new_Kd

    def repeatedly(self,current_value):
        '''
        Method that repeatedly runs the control algorithm. Compares
        setpoint to actual signal value. This error is multiplied by
        the proportional gain and sent to the plant.
        @param current_value: Received current value from feedback loop
        @return actuate_signal: % duty cycle sent to device driver
        '''
        #Define current value variable to be used in control algorithm
        self.__current_value = current_value
```

```
        self._error = self.setpoint - self.__current_value

        # Calculate integral of error (Esum)
        self._esum += self._error

        if self._esum > self.saturation:
            self._esum = self.saturation
        elif self._esum < -self.saturation:
            self._esum = -self.saturation

        #Calculate Derivative of Error
        self._deriv = self._error - self._perror
        self._perror = self._error

        #Creates actuation signal from the proportional gain mulitplied b
        self.__actuate_signal = self._error*self.Kp + self._esum*self.Ki+
        return self.__actuate_signal

    def percent_completion(self):
        ''' Returns the completion calculation for the controlled path.
        @return error The error from the desired position and current pos
        '''
        try:
            percent = (self._error/self.setpoint)*100
        except ZeroDivisionError:
            return 0
        else:
            return percent

    def clear_controller(self):
        ''' Clears the esum for a new target location.
        '''
        self._esum = 0
        self._error = 0
        self.__actuate_signal = 0
        self._perror = 0
        self._deriv = 0
```