```
from machine import Pin, I2C, Timer
from board import *
from bno055 import BNO055 # IMU

from drv8833 import DRV8833 # your implementation
from motor import PIDMotor # your implementation, make sure this is named right!
from encoder import Encoder # your implementation, don't forget clear_count
from balance import Balance

import gc # for garbage collection methods

i2c = I2C(0, sda=23, scl=22, freq=12500)
imu = BNO055(i2c)

accel = imu.accelerometer()
alpha = 90 - math.asin(accel/9.8)
actual = imu.euler()

print(alpha)
print(actual)

```

```
from machine import Pin, PWM

class DRV8833:

    def __init__(self, pinA, pinB, frequency=10000):
        '''Instantiate controller for one motor.
        pinA: pin connected to AIN1 or BIN1
        pinB: pin connected to AIN2 or BIN2
        frequency: pwm frequency
        '''
        self.pin1 = PWM(Pin(pinA), freq=frequency, timer=2)
        self.pin2 = PWM(Pin(pinB), freq=frequency, timer=3)

    def set_speed(self, value):
        if value > 100:
            value = 100
        elif value < -100:
            value = -100
        '''value: -100 ... 100 sets speed (duty cycle) and direction'''
        if value > 0:
            self.pin1.duty(100)
            self.pin2.duty(100 - value)
        elif value < 0:
            self.pin1.duty(100 - (-1 * value))
            self.pin2.duty(100)
        else:
            self.pin1.duty(value)
            self.pin2.duty(value)
```

```
from machine import Pin, DEC, PWM
from drv8833 import DRV8833
import time


class Encoder:

    def __init__(self, chA, chB, unit, counts_per_turn=24*75, wheel_diameter=330):
        '''Decode output from quadrature encoder connected to pins chA, chB.
        unit: DEC unit to use (0 ... 7).
        counts_per_turn: Number of counts per turn of the motor drive shaft. For scaling
cps to rpm.
        wheel_diameter: In [mm]. For scaling count to distance traveled.
        '''
        self.p1 = Pin(chA, mode=Pin.IN)
        self.p2 = Pin(chB, mode=Pin.IN)
        self.cpt = counts_per_turn
        self.dia = wheel_diameter
        self.dec = DEC(unit, self.p1, self.p2)
        self.count = self.dec.count()
        self.time = time.time()
        self.cps = 0

    def get_count(self):
        return self.dec.count()

    def get_distance(self):
        return get_count() / self.cpt * 3.14 * self.dia / 1000

    def get_cps(self):
        count = self.dec.count()
        curr_time = time.time()
        diff = count - self.count
        timediff = curr_time - self.time
        self.time = curr_time
        self.count = self.dec.count()
        self.cps = diff/timediff
        return self.cps

    def get_rpm(self):
        return self.get_cps()/self.cpt * 60

    def clear_count(self):
        # modify to match the variable names used in your code:
        self.dec.clear()
        self.count = self.dec.count()
        self.time = time.time()
```

```python
from drv8833 import DRV8833
from encoder import Encoder

class PIDMotor:

    def __init__(self, motor, encoder):
        '''Controller for a single motor
        motor: motor driver (DRV8833)
        encoder: motor encoder (Encoder)
        '''
        self.mot = motor
        self.end = encoder
        self.integ = 0

    def p_control(self, desired_cps, P=1):
        '''Set motor control to rotate at desired_cps'''
        actual_cps = self.end.get_cps()
        error = desired_cps - actual_cps
        self.mot.set_speed(P*error)
        # return speed (e.g. for plotting)
        return actual_cps

    def pi_control(self, desired_cps, Ts, P=1, I=1):
        actual_cps = self.end.get_cps()
        error = desired_cps - actual_cps
        self.integ += error * Ts/1000
        # clamp integrator, e.g. if desired_cps exceeds maximum motor speed
        self.integ = max(-150, min(self.integ, 150))
        self.mot.set_speed(P*error + I*self.integ)
        return actual_cps
```

```
24  _POWER_SUSPEND = const(0x02)
25
26
27  class BNO055:
28      """
29      Driver for the BNO055 9DOF IMU sensor.
30
31      Example::
32
33          import bno055
34          from machine import I2C, Pin
35
36          i2c = I2C(-1, Pin(5), Pin(4), timeout=1000)
37          s = bno055.BNO055(i2c)
38          print(s.temperature())
39          print(s.euler())
40      """
41
42      def __init__(self, i2c, address=0x28):
43          self.i2c = i2c
44          self.address = address
45          self.init()
46
47      def _registers(self, register, struct, value=None, scale=1):
48          if value is None:
49              size = ustruct.calcsize(struct)
50              data = self.i2c.readfrom_mem(self.address, register, size)
51              value = ustruct.unpack(struct, data)
52              if scale != 1:
53                  value = tuple(v * scale for v in value)
54              return value
55          if scale != 1:
56              value = tuple(v / scale for v in value)
57          data = ustruct.pack(struct, *value)
58          self.i2c.writeto_mem(self.address, register, data)
59
60      def _register(self, value=None, register=0x00, struct='B'):
61          if value is None:
62              return self._registers(register, struct=struct)[0]
63          self._registers(register, struct=struct, value=(value,))
64
65      _chip_id = partial(_register, register=0x00, value=None)
66      _power_mode = partial(_register, register=0x3e)
67      _system_trigger = partial(_register, register=0x3f)
68      _page_id = partial(_register, register=0x07)
69      operation_mode = partial(_register, register=0x3d)
70      temperature = partial(_register, register=0x34, value=None)
71      accelerometer = partial(_registers, register=0x08, struct='<hhh',
```

```python
def partial(func, *args, **kwargs):
    def _partial(*more_args, **more_kwargs):
        kw = kwargs.copy()
        kw.update(more_kwargs)
        return func(*(args + more_args), **kw)
    return _partial


def update_wrapper(wrapper, wrapped):
    # Dummy impl
    return wrapper


def wraps(wrapped):
    # Dummy impl
    return lambda x: x

def reduce(function, iterable, initializer=None):
    it = iter(iterable)
    if initializer is None:
        value = next(it)
    else:
        value = initializer
    for element in it:
        value = function(value, element)
    return value

```

```python
import gc

class Balance:
    radToDeg = 57.3 # radians to degrees, really just another scaling factor

    def __init__(self, lMotor, rMotor, imu, dt):
        self.pidL = lMotor
        self.pidR = rMotor
        self.imu = imu
        self.dt = dt

        # Working PID Constants
        self.kp = 219
        self.ki = 45

        self.mkp = 0.045
        self.mki = 0.5

        # the actual setpoint (takes into account position feedback)
        self.setPoint  = 0.07
        # the upward angle if at starting position (no position feedback)
        self.basePoint = 0.07
        self.balancing = False
        self.count = 0
        # integrator state
        self.integ = 0

    # set PI constants
    def set_balance_pi(self, p, i):
        self.kp = p
        self.ki = i

    def set_motor_pi(self, p, i):
        self.mkp = p
        self.mki = i

    # for keeping track of how long it has been balancing
    def increment_count(self):
        self.count += 1

    def do_balance(self):
        angle = (self.imu.euler()[2] - 90) / self.radToDeg
        print(angle)

        # if relatively straigt up
        if (abs(angle) < 0.1):
            # and has been held up for 3 seconds while not actively balancing
            if (self.count > 3 and not self.balancing):
```

```
from machine import Pin, I2C, Timer
from board import *
from bno055 import BNO055 # IMU

from drv8833 import DRV8833 # your implementation
from motor import PIDMotor # your implementation, make sure this is named right!
from encoder import Encoder # your implementation, don't forget clear_count
from balance import Balance

import gc # for garbage collection methods

# Setup motors
########## Check Pin Numbers! ##########
# Change pin numbers here to match yours or rewire your robot
leftEnc = Encoder(34, 39, 2)
leftM = DRV8833(19, 16)

rightEnc = Encoder(36, 4, 1)
rightM = DRV8833(17, 21)
########## Check Pin Numbers! ##########

###### If these don't work, choose your best PI values from the previous lab #####
# Feel free to experiment
mp = 0.045
mi = 0.5
###### If these don't work, choose your best PI values from the previous lab #####

# Balancing PI constants
bp = 219
bi = 45

# setup closed loop motor controllers
pidL = PIDMotor(leftM, leftEnc)
pidR = PIDMotor(rightM, rightEnc)

# setup IMU
i2c = I2C(0, sda=23, scl=22, freq=12500)
imu = BNO055(i2c)

# status LED
led = Pin(LED, mode=Pin.OUT)

dt = 0.02
ticks = 0
sec = 0
old_sec = 0
loopReady = False
```