VEX 機械人比賽訓練班
編程範例

```
def stopMoving():
    global pi, wheelRadius, wheelBase, digitalOutOff
    LF.set_stopping(HOLD)
    LM.set_stopping(HOLD)
    LB.set_stopping(HOLD)
    RF.set_stopping(HOLD)
    RM.set_stopping(HOLD)
    RB.set_stopping(HOLD)
    LF.stop()
    LM.stop()
    LB.stop()
    RF.stop()
    RM.stop()
    RB.stop()

def Driving_L_R(Driving_L_R__L, Driving_L_R__R):
    global pi, wheelRadius, wheelBase, digitalOutOff
    LF.set_velocity(Driving_L_R__L, PERCENT)
    LM.set_velocity(Driving_L_R__L, PERCENT)
    LB.set_velocity(Driving_L_R__L, PERCENT)
    RF.set_velocity(Driving_L_R__R, PERCENT)
    RM.set_velocity(Driving_L_R__R, PERCENT)
    RB.set_velocity(Driving_L_R__R, PERCENT)
    LF.spin(FORWARD)
    LM.spin(FORWARD)
    LB.spin(FORWARD)
    RF.spin(FORWARD)
    RM.spin(FORWARD)
    RB.spin(FORWARD)

def
DriveDistance_distance_mm__speed(DriveDistance_distance_mm__speed__distanc
e_mm_, DriveDistance_distance_mm__speed__speed):
    global pi, wheelRadius, wheelBase, digitalOutOff
    LF.set_velocity(DriveDistance_distance_mm__speed__speed, PERCENT)
    LM.set_velocity(DriveDistance_distance_mm__speed__speed, PERCENT)
    LB.set_velocity(DriveDistance_distance_mm__speed__speed, PERCENT)
    RF.set_velocity(DriveDistance_distance_mm__speed__speed, PERCENT)
    RM.set_velocity(DriveDistance_distance_mm__speed__speed, PERCENT)
    RB.set_velocity(DriveDistance_distance_mm__speed__speed, PERCENT)
    LF.spin_for(FORWARD, ((DriveDistance_distance_mm__speed__distance_mm_ /
wheelRadius) * 360), DEGREES, wait=False)
    LM.spin_for(FORWARD, ((DriveDistance_distance_mm__speed__distance_mm_
/ wheelRadius) * 360), DEGREES, wait=False)
    LB.spin_for(FORWARD, ((DriveDistance_distance_mm__speed__distance_mm_
/ wheelRadius) * 360), DEGREES, wait=False)
    RF.spin_for(FORWARD, ((DriveDistance_distance_mm__speed__distance_mm_
/ wheelRadius) * 360), DEGREES, wait=False)
    RM.spin_for(FORWARD, ((DriveDistance_distance_mm__speed__distance_mm_
/ wheelRadius) * 360), DEGREES, wait=False)
    RB.spin_for(FORWARD, ((DriveDistance_distance_mm__speed__distance_mm_
/ wheelRadius) * 360), DEGREES)

def Turning_angle_speed(Turning_angle_speed__angle,
Turning_angle_speed__speed):
    global pi, wheelRadius, wheelBase, digitalOutOff
    LF.set_velocity(Turning_angle_speed__speed, PERCENT)
    LM.set_velocity(Turning_angle_speed__speed, PERCENT)
    LB.set_velocity(Turning_angle_speed__speed, PERCENT)
    RF.set_velocity(Turning_angle_speed__speed, PERCENT)
    RM.set_velocity(Turning_angle_speed__speed, PERCENT)
    RB.set_velocity(Turning_angle_speed__speed, PERCENT)
    LF.spin_for(FORWARD, ((Turning_angle_speed__angle * wheelBase) /
wheelRadius), DEGREES, wait=False)
    LM.spin_for(FORWARD, ((Turning_angle_speed__angle * wheelBase) /
wheelRadius), DEGREES, wait=False)
    LB.spin_for(FORWARD, ((Turning_angle_speed__angle * wheelBase) /
wheelRadius), DEGREES, wait=False)
    RF.spin_for(REVERSE, ((Turning_angle_speed__angle * wheelBase) /
wheelRadius), DEGREES, wait=False)
    RM.spin_for(REVERSE, ((Turning_angle_speed__angle * wheelBase) /
wheelRadius), DEGREES, wait=False)
    RB.spin_for(REVERSE, ((Turning_angle_speed__angle * wheelBase) /
wheelRadius), DEGREES)

def ondriver_drivercontrol_0():
    global pi, wheelRadius, wheelBase, digitalOutOff
    # 初始化
    digital_out_a.set(False)
    digitalOutOff = True
    while True:
        if not controller_1.axis3.position() == 0 or not controller_1.axis1.position()
== 0:
            Driving_L_R(controller_1.axis3.position() + controller_1.axis1.position()
* 0.7, controller_1.axis3.position() - controller_1.axis1.position() * 0.7)
        else:
            stopMoving()
        wait(5, MSEC)

def onevent_controller_1buttonL1_pressed_0():
    global pi, wheelRadius, wheelBase, digitalOutOff
    digital_out_a.set(True)

def onevent_controller_1buttonA_pressed_0():
    global pi, wheelRadius, wheelBase, digitalOutOff
    # 氣動 Pneumatic
    if digitalOutOff:
        digital_out_a.set(True)
        digitalOutOff = False
        wait(0.1, SECONDS)
    else:
        digital_out_a.set(False)
        digitalOutOff = True
        wait(0.1, SECONDS)

def onevent_controller_1buttonL2_pressed_0():
    global pi, wheelRadius, wheelBase, digitalOutOff
    digital_out_a.set(False)

def onauton_autonomous_0():
    global pi, wheelRadius, wheelBase, digitalOutOff
    wheelRadius = (4 * 2.54) / 0.2
    pi = 3.141592654
    wheelBase = 110

# create a function for handling the starting and stopping of all autonomous tasks
def vexcode_auton_function():
    # Start the autonomous control tasks
    auton_task_0 = Thread( onauton_autonomous_0 )
    # wait for the driver control period to end
    while( competition.is_autonomous() and competition.is_enabled() ):
        # wait 10 milliseconds before checking again
        wait( 10, MSEC )
    # Stop the autonomous control tasks
    auton_task_0.stop()

def vexcode_driver_function():
    # Start the driver control tasks
    driver_control_task_0 = Thread( ondriver_drivercontrol_0 )

    # wait for the driver control period to end
    while( competition.is_driver_control() and competition.is_enabled() ):
        # wait 10 milliseconds before checking again
        wait( 10, MSEC )
    # Stop the driver control tasks
    driver_control_task_0.stop()


# register the competition functions
competition = Competition( vexcode_driver_function, vexcode_auton_function )

# system event handlers
controller_1.buttonL1.pressed(onevent_controller_1buttonL1_pressed_0)
controller_1.buttonA.pressed(onevent_controller_1buttonA_pressed_0)
controller_1.buttonL2.pressed(onevent_controller_1buttonL2_pressed_0)
# add 15ms delay to make sure events are registered correctly.
wait(15, MSEC)
```