ASSIGNMENT 4

NAME: ASWIN K U

TEAM ID: PNT2022TMID04593

PROGRAM:

```
import RPi.GPIO as GPIO
import MFRC522
import signal
from gpiozero import MCP3008, LED
from time import sleep


import boto3
import re
from boto3.dynamodb.conditions import Key, Attr
import time


#  ============================================================
#  Functions
#  ============================================================
# turn off the green light and turn on the red light.
def occupied():
GPIO.output(5, GPIO.HIGH)
GPIO.output(6, GPIO.LOW)



# turn off the red light and turn on the green light.
def available():
GPIO.output(6, GPIO.HIGH)
GPIO.output(5, GPIO.LOW)
```

```
# check if the lot is taken
def check_current_holder():
    response = Parking_Lots_Table.scan(
FilterExpression=Attr('ID').contains(Lot_Scanner) &Attr('Carpark_Name').contains(Carpark)
    )

    item = response['Items']

    distance, current_holder, ID = str(item).split(',', 2)
ch, actual_holder = str(current_holder).split(':', 1)
actual_holder = re.sub("['u ]", '', actual_holder)
    return actual_holder


# check current number of available lots
def check_available_lots():
    response = Carparks_Table.scan(
FilterExpression=Attr('Carpark_Name').contains(Carpark)
    )

    item = response['Items']

carpark_name, available_lots = str(item).split(',', 1)
    al, actual_number_of_lots = str(available_lots).split(':', 1)
actual_number_of_lots = re.sub("['u}\] ]", '', actual_number_of_lots)
    return actual_number_of_lots


# Capture SIGINT for cleanup when the script is aborted
def end_read(signal, frame):
```

```
    global continue_reading
    print "Ctrl+C captured, ending read."
continue_reading = False
GPIO.cleanup()



#  ================================================================
#  Variable Declarations
#  ================================================================
uid = None
continue_reading = True
Lot_Scanner = "C1"
Carpark = "Bukit Batok Carpark"

# AWS Credentials
access_key = "AKIAXXXXXXXXXXXXXXXX"
secret_access_key = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

# Hook the SIGINT
signal.signal(signal.SIGINT, end_read)

# Create an object of the class MFRC522
mfrc522 = MFRC522.MFRC522()

# Welcome message
print("This is Lot {}".format(Lot_Scanner))
print "Press Ctrl-C to stop."

dynamodb = boto3.resource('dynamodb', aws_access_key_id=access_key,
aws_secret_access_key=secret_access_key, region_name='us-west-2')
Parking_Lots_Table = dynamodb.Table('Parking_Lots')
```

```
Carparks_Table = dynamodb.Table('Carparks')
timeout = None

adc = MCP3008(channel=0)

GPIO.setwarnings(False)
Occupied = GPIO.setup(5,GPIO.OUT)
Available = GPIO.setup(6,GPIO.OUT)

# Set light to green at the start.
available()

#  ================================================================
#  Main
#  ================================================================
if __name__ == "__main__":
   while continue_reading:
      # check if the B1 lot is taken
actual_holder = check_current_holder()

      # check current number of available lots
actual_number_of_lots = check_available_lots()

      # Scan for cards
      (status, TagType) = mfrc522.MFRC522_Request(mfrc522.PICC_REQIDL)

      # Check light sensor value
light_value = adc.value

      # If a card is found
      if status == mfrc522.MI_OK and light_value> 0.62:
```

```
        # Get the UID of the card
        (status,uid) = mfrc522.MFRC522_Anticoll()
print("Car detected! Its RFID card's UID is {}".format(uid))

        timeout = time.time() + 20

        if actual_holder == 'None':
            # update parking lots table from none to uid
Parking_Lots_Table.update_item(
                Key={
                    'ID': Lot_Scanner,
                    'Carpark_Name': Carpark
                },
UpdateExpression='SET Current_Holder= :val',
ExpressionAttributeValues={
':val': str(uid)
                }
            )

            # reduce available lots in carparks table by 1
Carparks_Table.update_item(
                Key={
                    'Carpark_Name': Carpark
                },
UpdateExpression='SET Available_Lots= :val',
ExpressionAttributeValues={
':val': str(int(actual_number_of_lots) - 1)
                }
            )
print('Updated DB to UID')
occupied()
```

```
            else:
print("DB already updated")
sleep(10)


        if time.time() > timeout:
            if actual_holder != 'None':
                # update parking lots table from uid to none
Parking_Lots_Table.update_item(
                    Key={
                        'ID': Lot_Scanner,
                        'Carpark_Name': Carpark
                    },
UpdateExpression='SET Current_Holder= :val',
ExpressionAttributeValues={
':val': 'None'
                    }
                )


                # increase available lots in carparks table by 1
Carparks_Table.update_item(
                    Key={
                        'Carpark_Name': Carpark
                    },
UpdateExpression='SET Available_Lots= :val',
ExpressionAttributeValues={
':val': str(int(actual_number_of_lots) + 1)
                    }
                )
print("Car not detected after 20 seconds from last detection")
print('Updated DB to None')
available()
```