Tempexample.py

```
import RPi.GPIO as GPIO
import temp
import time

# initialize GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
#GPIO.cleanup()

# read data using Pin GPIO21
instance = dht11.DHT11(pin=21)

while True:
    result = instance.read()
    if result.is_valid():
        print("Temp: %d C" % result.temperature +' '+"Humid: %d %%" %
result.humidity)

    time.sleep(1)
```

temp.py

```
import time
import RPi


class tempResult:
    'temp sensor result returned by temp.read() method'

    ERR_NO_ERROR = 0
    ERR_MISSING_DATA = 1
    ERR_CRC = 2

    error_code = ERR_NO_ERROR
    temperature = -1
    humidity = -1

    def __init__(self, error_code, temperature, humidity):
        self.error_code = error_code
        self.temperature = temperature
        self.humidity = humidity

    def is_valid(self):
        return self.error_code == tempResult.ERR_NO_ERROR


class temp:
    'temp sensor reader class for Raspberry'

    __pin = 0
```

```
    def __init__(self, pin):
        self.__pin = pin

    def read(self):
        RPi.GPIO.setup(self.__pin, RPi.GPIO.OUT)

        # send initial high
        self.__send_and_sleep(RPi.GPIO.HIGH, 0.05)

        # pull down to low
        self.__send_and_sleep(RPi.GPIO.LOW, 0.02)

        # change to input using pull up
        RPi.GPIO.setup(self.__pin, RPi.GPIO.IN, RPi.GPIO.PUD_UP)

        # collect data into an array
        data = self.__collect_input()

        # parse lengths of all data pull up periods
        pull_up_lengths = self.__parse_data_pull_up_lengths(data)

        # if bit count mismatch, return error (4 byte data + 1 byte
checksum)
        if len(pull_up_lengths) != 40:
            return DHT11Result(DHT11Result.ERR_MISSING_DATA, 0, 0)

        # calculate bits from lengths of the pull up periods
        bits = self.__calculate_bits(pull_up_lengths)

        # we have the bits, calculate bytes
        the_bytes = self.__bits_to_bytes(bits)

        # calculate checksum and check
        checksum = self.__calculate_checksum(the_bytes)
        if the_bytes[4] != checksum:
            return DHT11Result(DHT11Result.ERR_CRC, 0, 0)

        # ok, we have valid data, return it
        return DHT11Result(DHT11Result.ERR_NO_ERROR, the_bytes[2],
the_bytes[0])

    def __send_and_sleep(self, output, sleep):
        RPi.GPIO.output(self.__pin, output)
        time.sleep(sleep)

    def __collect_input(self):
        # collect the data while unchanged found
        unchanged_count = 0

        # this is used to determine where is the end of the data
        max_unchanged_count = 100

        last = -1
        data = []
```

```
        while True:
            current = RPi.GPIO.input(self.__pin)
            data.append(current)
            if last != current:
                unchanged_count = 0
                last = current
            else:
                unchanged_count += 1
                if unchanged_count > max_unchanged_count:
                    break

        return data

    def __parse_data_pull_up_lengths(self, data):
        STATE_INIT_PULL_DOWN = 1
        STATE_INIT_PULL_UP = 2
        STATE_DATA_FIRST_PULL_DOWN = 3
        STATE_DATA_PULL_UP = 4
        STATE_DATA_PULL_DOWN = 5

        state = STATE_INIT_PULL_DOWN

        lengths = [] # will contain the lengths of data pull up periods
        current_length = 0 # will contain the length of the previous
period

        for i in range(len(data)):

            current = data[i]
            current_length += 1

            if state == STATE_INIT_PULL_DOWN:
                if current == RPi.GPIO.LOW:
                    # ok, we got the initial pull down
                    state = STATE_INIT_PULL_UP
                    continue
                else:
                    continue
            if state == STATE_INIT_PULL_UP:
                if current == RPi.GPIO.HIGH:
                    # ok, we got the initial pull up
                    state = STATE_DATA_FIRST_PULL_DOWN
                    continue
                else:
                    continue
            if state == STATE_DATA_FIRST_PULL_DOWN:
                if current == RPi.GPIO.LOW:
                    # we have the initial pull down, the next will be the
data pull up
                    state = STATE_DATA_PULL_UP
                    continue
                else:
                    continue
            if state == STATE_DATA_PULL_UP:
```

```
                if current == RPi.GPIO.HIGH:
                    # data pulled up, the length of this pull up will
determine whether it is 0 or 1
                    current_length = 0
                    state = STATE_DATA_PULL_DOWN
                    continue
                else:
                    continue
            if state == STATE_DATA_PULL_DOWN:
                if current == RPi.GPIO.LOW:
                    # pulled down, we store the length of the previous
pull up period
                    lengths.append(current_length)
                    state = STATE_DATA_PULL_UP
                    continue
                else:
                    continue

        return lengths

    def __calculate_bits(self, pull_up_lengths):
        # find shortest and longest period
        shortest_pull_up = 1000
        longest_pull_up = 0

        for i in range(0, len(pull_up_lengths)):
            length = pull_up_lengths[i]
            if length < shortest_pull_up:
                shortest_pull_up = length
            if length > longest_pull_up:
                longest_pull_up = length

        # use the halfway to determine whether the period it is long or
short
        halfway = shortest_pull_up + (longest_pull_up - shortest_pull_up)
/ 2
        bits = []

        for i in range(0, len(pull_up_lengths)):
            bit = False
            if pull_up_lengths[i] > halfway:
                bit = True
            bits.append(bit)

        return bits

    def __bits_to_bytes(self, bits):
        the_bytes = []
        byte = 0

        for i in range(0, len(bits)):
            byte = byte << 1
            if (bits[i]):
                byte = byte | 1
```

```
            else:
                byte = byte | 0
            if ((i + 1) % 8 == 0):
                the_bytes.append(byte)
                byte = 0

        return the_bytes

    def __calculate_checksum(self, the_bytes):
        return the_bytes[0] + the_bytes[1] + the_bytes[2] + the_bytes[3]
& 255
```