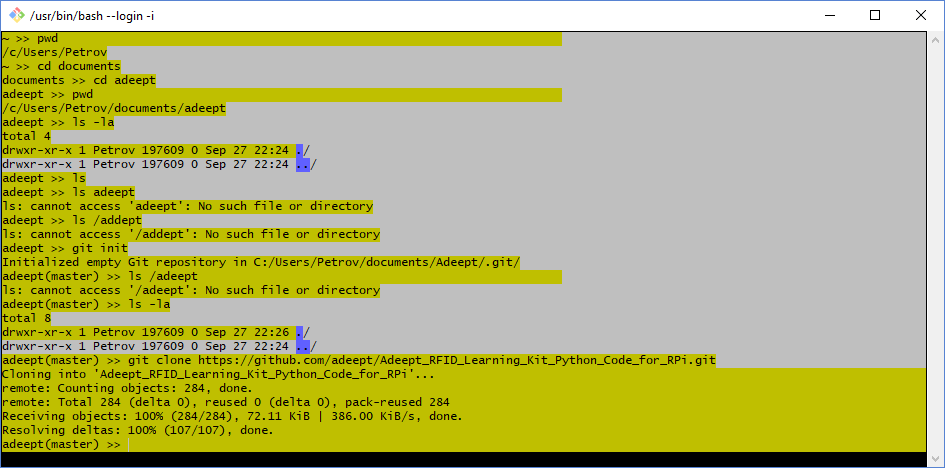
**PYTHON PROGRAMMING ON RASPBERRY PI - by Andre Petrov**

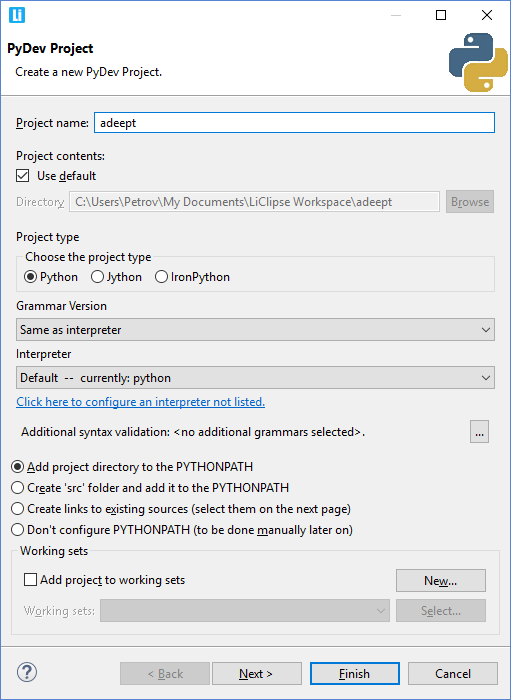
The following steps were taken to test Adeept’ s wiring kit and Python code on Raspberry Pi.

1. Created local Git working directory ‘adeept’, initialised it and cloned Addept repo from GitHub. NOTE: There was no intention to modify and upload Adeept code.



**=============================**

1. Opened a new Python project called ‘adeept’ in Eclipse IDE on Windows and created a working environment using files cloned from GitHub



**=============================**

**3. Downloaded Git to Raspberry Pi**

*Sudo apt-get install git*

*Git init*

Directory set at **/home/pi/.git**

|  |  |  |
| --- | --- | --- |
| **20171009_211714** | **20171009_211837** |  |

**=============================**

**4. Downloaded Adeept’s Python code to RPi**

*$ git clone https://github.com/adeept/Adeept\_RFID\_Learning\_Kit\_Python\_Code\_for\_RPi.git*

**=============================**

1. **Installed WiringPI on RPi - C code library (Did not really need to…)**

*git clone git://git.drogon.net/wiringPi*

*Cloning into 'wiringPi'...*

*remote: Counting objects: 1151, done.*

*remote: Compressing objects: 100% (957/957), done.*

*remote: Total 1151 (delta 804), reused 212 (delta 142)*

*Receiving objects: 100% (1151/1151), 364.87 KiB | 200.00 KiB/s, done.*

*Resolving deltas: 100% (804/804), done.*

***Install wiringPi:***

Step 1 : Get the source code

*$ git clone git://git.drogon.net/wiringPi*

*git clone git://git.drogon.net/wiringPi*

*Cloning into 'wiringPi'...*

*remote: Counting objects: 1151, done.*

*remote: Compressing objects: 100% (957/957), done.*

*remote: Total 1151 (delta 804), reused 212 (delta 142)*

*Receiving objects: 100% (1151/1151), 364.87 KiB | 200.00 KiB/s, done.*

*Resolving deltas: 100% (804/804), done.*

Step 2 : Compile and install

*$ cd wiringPi*

*$ git pull origin*

*$ sudo ./build*

Step 3: Checking whether GPIO was installed

*pi@raspberrypi:~/WiringPi $ sudo gpio -v*

gpio version: 2.44

Copyright (c) 2012-2017 Gordon Henderson

This is free software with ABSOLUTELY NO WARRANTY.

For details type: gpio -warranty

Raspberry Pi Details:

Type: Pi 3, Revision: 02, Memory: 1024MB, Maker: Sony

\* Device tree is enabled.

\*--> Raspberry Pi 3 Model B Rev 1.2

\* This Raspberry Pi supports user-level GPIO access.

**6. Install Python on RPI**

<https://www.raspberrypi.org/documentation/linux/software/python.md>

<<<Python 3 already installed on RPI>>

**=============================**

**7. Install Eclipse IDE on RPi**

$ *sudo apt-get install eclipse*

SUCCESS ! BUT MAY NEED TO CHANGE OpenJDK to Java 8 JDK

|  |
| --- |
| 20171008_174718 |

**=============================**

**8. Install Vim editor on RPi**

*git clone <https://github.com/vim/vim.git>*

*sudo apt-get install vim-gnome*

|  |  |  |  |
| --- | --- | --- | --- |
| 20171009_220319 | 20171009_220937 | 20171009_221502 |  |

<<<Not using it really, launching Eclipse to work with Python code on RPi>>>

1. **Download Adeept code to RPi**

*git clone https://github.com/adeept/Adeept\_RFID\_Learning\_Kit\_Python\_Code\_for\_RPi*

|  |  |  |
| --- | --- | --- |
| 20171013_095611 |  |  |

**10. Build circuits and run code on RPi via VNC**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **20171009_223206** | **20171009_235325** | **20171010_161314** | **20171010_210836** |  |
|  |  |  |  |  |
| **20171010_211425** | **20171010_224726** | **20171010_230604** | **20171012_122402** |  |
| **20171011_091704** | **20171011_083158** | **20171012_134842** | **20171012_155733** |  |
|  |  |  |  |  |
| **20171012_165231** | **20171012_194743** | **20171014_183655** |  |  |
|  |  |  |  |  |

1. **Encountered problems / issues with Adeept code and kit**

**11.1 Lesson 19 - RFID**

<<PROBLEM - when running Read.py it tries to import MFRC522…..

#!/usr/bin/env python

# -\*- coding: utf8 -\*-

import RPi.GPIO as GPIO

import MFRC522

import signal

continue\_reading = True

# 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()

# Hook the SIGINT

signal.signal(signal.SIGINT, end\_read)

# Create an object of the class MFRC522

MIFAREReader = MFRC522.MFRC522()

# Welcome message

print *"Welcome to the MFRC522 data read example"*

print *"Press Ctrl-C to stop."*

# This loop keeps checking for chips. If one is near it will get the UID and authenticate

while continue\_reading:

# Scan for cards

(status,TagType) = MIFAREReader.MFRC522\_Request(MIFAREReader.PICC\_REQIDL)

# If a card is found

if status == MIFAREReader.MI\_OK:

print *"Card detected"*

# Get the UID of the card

(status,uid) = MIFAREReader.MFRC522\_Anticoll()

# If we have the UID, continue

if status == MIFAREReader.MI\_OK:

# Print UID

print *"Card read UID: "*+str(uid[0])+*","*+str(uid[1])+*","*+str(uid[2])+*","*+str(uid[3])

# This is the default key for authentication

key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]

# Select the scanned tag

MIFAREReader.MFRC522\_SelectTag(uid)

# Authenticate

status = MIFAREReader.MFRC522\_Auth(MIFAREReader.PICC\_AUTHENT1A, 8, key, uid)

# Check if authenticated

if status == MIFAREReader.MI\_OK:

MIFAREReader.MFRC522\_Read(8)

MIFAREReader.MFRC522\_StopCrypto1()

else:

print *"Authentication error"*

>>>>>>>Following that, MFRC tries to import spi

#!/usr/bin/env python

# -\*- coding: utf8 -\*-

import RPi.GPIO as GPIO

import spi

import signal

import time

class **MFRC522**:

NRSTPD = 22

MAX\_LEN = 16

PCD\_IDLE = 0x00

PCD\_AUTHENT = 0x0E

PCD\_RECEIVE = 0x08

PCD\_TRANSMIT = 0x04

PCD\_TRANSCEIVE = 0x0C

PCD\_RESETPHASE = 0x0F

PCD\_CALCCRC = 0x03

PICC\_REQIDL = 0x26

PICC\_REQALL = 0x52

PICC\_ANTICOLL = 0x93

PICC\_SElECTTAG = 0x93

PICC\_AUTHENT1A = 0x60

PICC\_AUTHENT1B = 0x61

PICC\_READ = 0x30

PICC\_WRITE = 0xA0

PICC\_DECREMENT = 0xC0

PICC\_INCREMENT = 0xC1

PICC\_RESTORE = 0xC2

PICC\_TRANSFER = 0xB0

PICC\_HALT = 0x50

MI\_OK = 0

MI\_NOTAGERR = 1

MI\_ERR = 2

Reserved00 = 0x00

CommandReg = 0x01

CommIEnReg = 0x02

DivlEnReg = 0x03

CommIrqReg = 0x04

DivIrqReg = 0x05

ErrorReg = 0x06

Status1Reg = 0x07

Status2Reg = 0x08

FIFODataReg = 0x09

FIFOLevelReg = 0x0A

WaterLevelReg = 0x0B

ControlReg = 0x0C

BitFramingReg = 0x0D

CollReg = 0x0E

Reserved01 = 0x0F

Reserved10 = 0x10

ModeReg = 0x11

TxModeReg = 0x12

RxModeReg = 0x13

TxControlReg = 0x14

TxAutoReg = 0x15

TxSelReg = 0x16

RxSelReg = 0x17

RxThresholdReg = 0x18

DemodReg = 0x19

Reserved11 = 0x1A

Reserved12 = 0x1B

MifareReg = 0x1C

Reserved13 = 0x1D

Reserved14 = 0x1E

SerialSpeedReg = 0x1F

Reserved20 = 0x20

CRCResultRegM = 0x21

CRCResultRegL = 0x22

Reserved21 = 0x23

ModWidthReg = 0x24

Reserved22 = 0x25

RFCfgReg = 0x26

GsNReg = 0x27

CWGsPReg = 0x28

ModGsPReg = 0x29

TModeReg = 0x2A

TPrescalerReg = 0x2B

TReloadRegH = 0x2C

TReloadRegL = 0x2D

TCounterValueRegH = 0x2E

TCounterValueRegL = 0x2F

Reserved30 = 0x30

TestSel1Reg = 0x31

TestSel2Reg = 0x32

TestPinEnReg = 0x33

TestPinValueReg = 0x34

TestBusReg = 0x35

AutoTestReg = 0x36

VersionReg = 0x37

AnalogTestReg = 0x38

TestDAC1Reg = 0x39

TestDAC2Reg = 0x3A

TestADCReg = 0x3B

Reserved31 = 0x3C

Reserved32 = 0x3D

Reserved33 = 0x3E

Reserved34 = 0x3F

serNum = []

def **\_\_init\_\_**(*self*, dev=*'/dev/spidev0.0'*, spd=1000000):

spi.openSPI(device=dev,speed=spd)

GPIO.setmode(GPIO.BOARD)

GPIO.setup(22, GPIO.OUT)

GPIO.output(*self*.NRSTPD, 1)

*self*.MFRC522\_Init()

def **MFRC522\_Reset**(*self*):

*self*.Write\_MFRC522(*self*.CommandReg, *self*.PCD\_RESETPHASE)

def **Write\_MFRC522**(*self*, addr, val):

spi.transfer(((addr<<1)&0x7E,val))

def **Read\_MFRC522**(*self*, addr):

val = spi.transfer((((addr<<1)&0x7E) | 0x80,0))

return val[1]

def **SetBitMask**(*self*, reg, mask):

tmp = *self*.Read\_MFRC522(reg)

*self*.Write\_MFRC522(reg, tmp | mask)

def **ClearBitMask**(*self*, reg, mask):

tmp = *self*.Read\_MFRC522(reg);

*self*.Write\_MFRC522(reg, tmp & (~mask))

def **AntennaOn**(*self*):

temp = *self*.Read\_MFRC522(*self*.TxControlReg)

if(~(temp & 0x03)):

*self*.SetBitMask(*self*.TxControlReg, 0x03)

def **AntennaOff**(*self*):

*self*.ClearBitMask(*self*.TxControlReg, 0x03)

def **MFRC522\_ToCard**(*self*,command,sendData):

backData = []

backLen = 0

status = *self*.MI\_ERR

irqEn = 0x00

waitIRq = 0x00

lastBits = None

n = 0

i = 0

if command == *self*.PCD\_AUTHENT:

irqEn = 0x12

waitIRq = 0x10

if command == *self*.PCD\_TRANSCEIVE:

irqEn = 0x77

waitIRq = 0x30

*self*.Write\_MFRC522(*self*.CommIEnReg, irqEn|0x80)

*self*.ClearBitMask(*self*.CommIrqReg, 0x80)

*self*.SetBitMask(*self*.FIFOLevelReg, 0x80)

*self*.Write\_MFRC522(*self*.CommandReg, *self*.PCD\_IDLE);

while(i<len(sendData)):

*self*.Write\_MFRC522(*self*.FIFODataReg, sendData[i])

i = i+1

*self*.Write\_MFRC522(*self*.CommandReg, command)

if command == *self*.PCD\_TRANSCEIVE:

*self*.SetBitMask(*self*.BitFramingReg, 0x80)

i = 2000

while True:

n = *self*.Read\_MFRC522(*self*.CommIrqReg)

i = i - 1

if ~((i!=0) and ~(n&0x01) and ~(n&waitIRq)):

break

*self*.ClearBitMask(*self*.BitFramingReg, 0x80)

if i != 0:

if (*self*.Read\_MFRC522(*self*.ErrorReg) & 0x1B)==0x00:

status = *self*.MI\_OK

if n & irqEn & 0x01:

status = *self*.MI\_NOTAGERR

if command == *self*.PCD\_TRANSCEIVE:

n = *self*.Read\_MFRC522(*self*.FIFOLevelReg)

lastBits = *self*.Read\_MFRC522(*self*.ControlReg) & 0x07

if lastBits != 0:

backLen = (n-1)\*8 + lastBits

else:

backLen = n\*8

if n == 0:

n = 1

if n > *self*.MAX\_LEN:

n = *self*.MAX\_LEN

i = 0

while i<n:

backData.append(*self*.Read\_MFRC522(*self*.FIFODataReg))

i = i + 1;

else:

status = *self*.MI\_ERR

return (status,backData,backLen)

def **MFRC522\_Request**(*self*, reqMode):

status = None

backBits = None

TagType = []

*self*.Write\_MFRC522(*self*.BitFramingReg, 0x07)

TagType.append(reqMode);

(status,backData,backBits) = *self*.MFRC522\_ToCard(*self*.PCD\_TRANSCEIVE, TagType)

if ((status != *self*.MI\_OK) | (backBits != 0x10)):

status = *self*.MI\_ERR

return (status,backBits)

def **MFRC522\_Anticoll**(*self*):

backData = []

serNumCheck = 0

serNum = []

*self*.Write\_MFRC522(*self*.BitFramingReg, 0x00)

serNum.append(*self*.PICC\_ANTICOLL)

serNum.append(0x20)

(status,backData,backBits) = *self*.MFRC522\_ToCard(*self*.PCD\_TRANSCEIVE,serNum)

if(status == *self*.MI\_OK):

i = 0

if len(backData)==5:

while i<4:

serNumCheck = serNumCheck ^ backData[i]

i = i + 1

if serNumCheck != backData[i]:

status = *self*.MI\_ERR

else:

status = *self*.MI\_ERR

return (status,backData)

def **CalulateCRC**(*self*, pIndata):

*self*.ClearBitMask(*self*.DivIrqReg, 0x04)

*self*.SetBitMask(*self*.FIFOLevelReg, 0x80);

i = 0

while i<len(pIndata):

*self*.Write\_MFRC522(*self*.FIFODataReg, pIndata[i])

i = i + 1

*self*.Write\_MFRC522(*self*.CommandReg, *self*.PCD\_CALCCRC)

i = 0xFF

while True:

n = *self*.Read\_MFRC522(*self*.DivIrqReg)

i = i - 1

if not ((i != 0) and not (n&0x04)):

break

pOutData = []

pOutData.append(*self*.Read\_MFRC522(*self*.CRCResultRegL))

pOutData.append(*self*.Read\_MFRC522(*self*.CRCResultRegM))

return pOutData

def **MFRC522\_SelectTag**(*self*, serNum):

backData = []

buf = []

buf.append(*self*.PICC\_SElECTTAG)

buf.append(0x70)

i = 0

while i<5:

buf.append(serNum[i])

i = i + 1

pOut = *self*.CalulateCRC(buf)

buf.append(pOut[0])

buf.append(pOut[1])

(status, backData, backLen) = *self*.MFRC522\_ToCard(*self*.PCD\_TRANSCEIVE, buf)

if (status == *self*.MI\_OK) and (backLen == 0x18):

print *"Size: "* + str(backData[0])

return backData[0]

else:

return 0

def **MFRC522\_Auth**(*self*, authMode, BlockAddr, Sectorkey, serNum):

buff = []

# First byte should be the authMode (A or B)

buff.append(authMode)

# Second byte is the trailerBlock (usually 7)

buff.append(BlockAddr)

# Now we need to append the authKey which usually is 6 bytes of 0xFF

i = 0

while(i < len(Sectorkey)):

buff.append(Sectorkey[i])

i = i + 1

i = 0

# Next we append the first 4 bytes of the UID

while(i < 4):

buff.append(serNum[i])

i = i +1

# Now we start the authentication itself

(status, backData, backLen) = *self*.MFRC522\_ToCard(*self*.PCD\_AUTHENT,buff)

# Check if an error occurred

if not(status == *self*.MI\_OK):

print *"AUTH ERROR!!"*

if not (*self*.Read\_MFRC522(*self*.Status2Reg) & 0x08) != 0:

print *"AUTH ERROR(status2reg & 0x08) != 0"*

# Return the status

return status

def **MFRC522\_StopCrypto1**(*self*):

*self*.ClearBitMask(*self*.Status2Reg, 0x08)

def **MFRC522\_Read**(*self*, blockAddr):

recvData = []

recvData.append(*self*.PICC\_READ)

recvData.append(blockAddr)

pOut = *self*.CalulateCRC(recvData)

recvData.append(pOut[0])

recvData.append(pOut[1])

(status, backData, backLen) = *self*.MFRC522\_ToCard(*self*.PCD\_TRANSCEIVE, recvData)

if not(status == *self*.MI\_OK):

print *"Error while reading!"*

i = 0

if len(backData) == 16:

print *"Sector "*+str(blockAddr)+*" "*+str(backData)

def **MFRC522\_Write**(*self*, blockAddr, writeData):

buff = []

buff.append(*self*.PICC\_WRITE)

buff.append(blockAddr)

crc = *self*.CalulateCRC(buff)

buff.append(crc[0])

buff.append(crc[1])

(status, backData, backLen) = *self*.MFRC522\_ToCard(*self*.PCD\_TRANSCEIVE, buff)

if not(status == *self*.MI\_OK) or not(backLen == 4) or not((backData[0] & 0x0F) == 0x0A):

status = *self*.MI\_ERR

print str(backLen)+*" backdata &0x0F == 0x0A "*+str(backData[0]&0x0F)

if status == *self*.MI\_OK:

i = 0

buf = []

while i < 16:

buf.append(writeData[i])

i = i + 1

crc = *self*.CalulateCRC(buf)

buf.append(crc[0])

buf.append(crc[1])

(status, backData, backLen) = *self*.MFRC522\_ToCard(*self*.PCD\_TRANSCEIVE,buf)

if not(status == *self*.MI\_OK) or not(backLen == 4) or not((backData[0] & 0x0F) == 0x0A):

print *"Error while writing"*

if status == *self*.MI\_OK:

print *"Data written"*

def **MFRC522\_DumpClassic1K**(*self*, key, uid):

i = 0

while i < 64:

status = *self*.MFRC522\_Auth(*self*.PICC\_AUTHENT1A, i, key, uid)

# Check if authenticated

if status == *self*.MI\_OK:

*self*.MFRC522\_Read(i)

else:

print *"Authentication error"*

i = i+1

def **MFRC522\_Init**(*self*):

GPIO.output(*self*.NRSTPD, 1)

*self*.MFRC522\_Reset();

*self*.Write\_MFRC522(*self*.TModeReg, 0x8D)

*self*.Write\_MFRC522(*self*.TPrescalerReg, 0x3E)

*self*.Write\_MFRC522(*self*.TReloadRegL, 30)

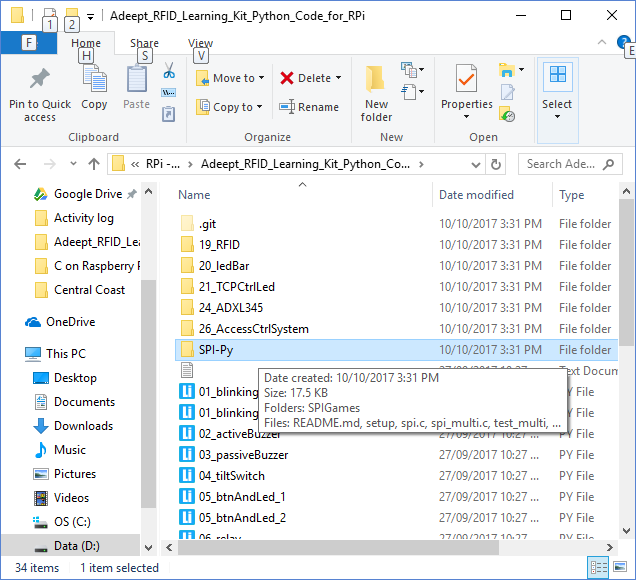
*self*.Write\_MFRC522(*self*.TReloadRegH, 0)

*self*.Write\_MFRC522(*self*.TxAutoReg, 0x40)

*self*.Write\_MFRC522(*self*.ModeReg, 0x3D)

*self*.AntennaOn()

>>>RFID kit contains a folder named SPI-PY



>>>> MFRC522 gives an error:

Import spi

ImportError: no module named spi

Traceback (most recent call last):

File "/home/pi/workspace/Adeept/19\_RFID/MFRC522-python/Read.py", line 5, in <module>

import MFRC522

File "/home/pi/workspace/Adeept/19\_RFID/MFRC522-python/MFRC522.py", line 5, in <module>

import spi

ImportError: No module named 'spi'

>>>>Checked the code downloaded from Git and found folder SPI but appears to be written in C<<<

Can I run this code in C????

|  |  |  |
| --- | --- | --- |
| 20171012_214053 | 20171012_214058 | 20171013_131452 |

|  |  |  |
| --- | --- | --- |
| 20171013_132346 | 20171013_133208 |  |

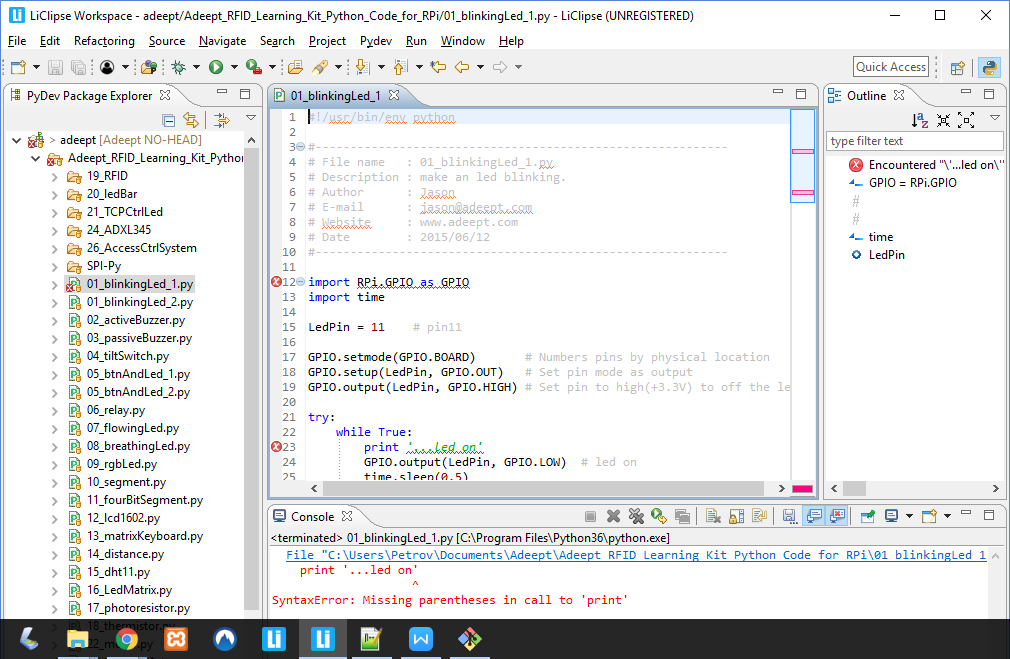
>>> solution - Code is written in Python 2 but I was trying to run it in IDE running Python 3.6.2

|  |  |  |
| --- | --- | --- |
| pip3_install_spi |  |  |

**Lesson 20**

Issues with code

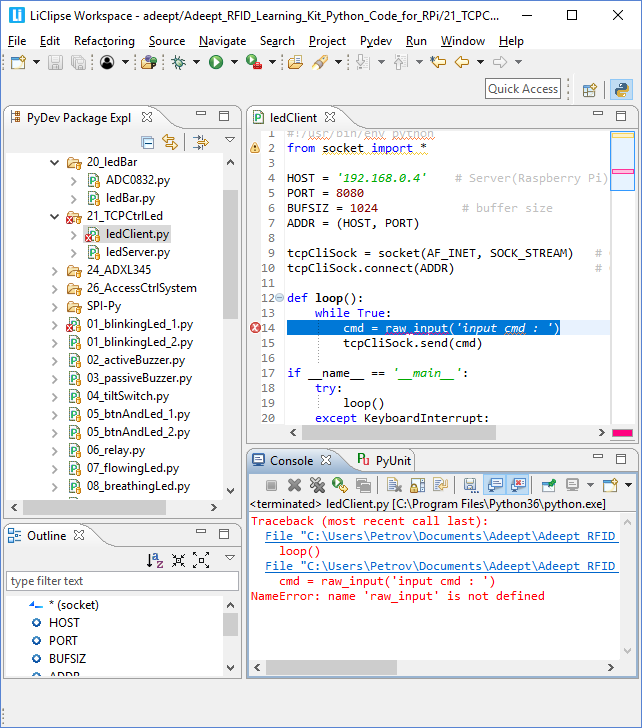
|  |  |  |
| --- | --- | --- |
| 20171012_214053 | integer error |  |
|  |  |  |



>>> Missing parenthesis in call to print

>>>float object cannot be interpreted as integer

**Lesson 21 - TCP /IP**



Replaced raw\_input( with raw\_input (

def **loop**():

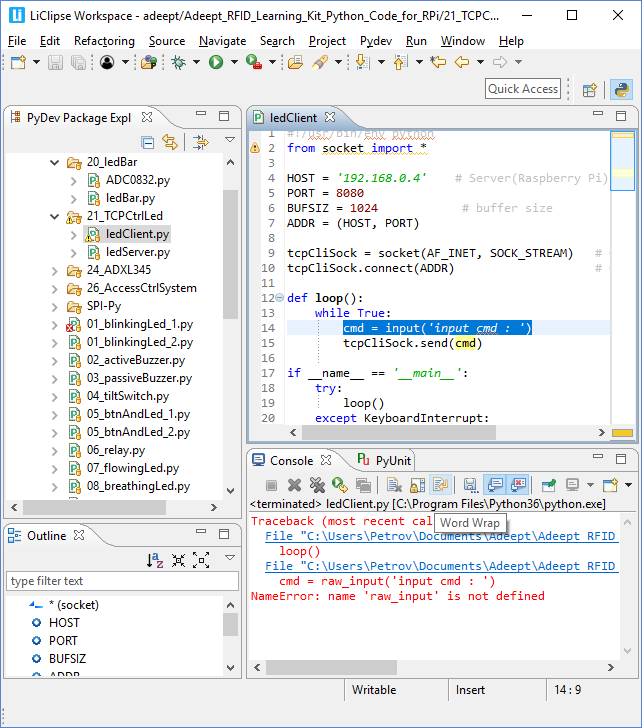
while True:

cmd = raw\_input (*'input cmd : '*)

output = cmd

tcpCliSock.send(output.encode(*'utf-8'*))

And added output and encoding, error went away but the led was not ‘on’ and ‘off’



The exercise 21 setup had the Server python program running on RPi and Client python program running on a Windows machine, using Python2.7 executable.  
  
While Client showed no errors while sending data, no data waere actually been sent until terminating the Client program.  
After  terminating the Client program, all unsent data was sent and received by Server all at once, and some socket exceptionr was raised on Server.  
  
After switching Windows machine to Mac machine, everything worked as expected, messages passing on time, LED turning on and off.