**Teach-In 2014 with Raspberry Pi: Part 6**

by Mike and Richard Tooley

These are text files of the source code listings printed in EPE.

They appear in the same order as in the articles.

Separate listings are split by four empty lines.

Snum\_str = input("Enter a numeric value: ")

value = int(num\_str)

# Test input

print(value)

Enter a numeric value: 123.0

Traceback (most recent call last):

File "C:/Python33/data\_validation\_4.py", line 3, in <module>

value = int(num\_str)

ValueError: invalid literal for int() with base 10: '123.0'

Enter a numeric value: 123

Traceback (most recent call last):

File "C:/Python33/data\_validation\_4.py", line 3, in <module>

value = int(num\_str)

ValueError: invalid literal for int() with base 10: ' 123'

Enter a numeric value: a123

Traceback (most recent call last):

File "C:/Python33/data\_validation\_4.py", line 3, in <module>

value = int(num\_str)

ValueError: invalid literal for int() with base 10: 'a123'

def get\_number(prompt = "Enter a numeric value: "):

while True:

# First remove any leading or trailing whitespace

num\_str = input(prompt).strip()

# make sure that all char can be in a typical number

if all(c in '+-.0123456789' for c in num\_str):

break

else:

print('Invalid character - try again!')

# Check whether integer or float has been input

if '.' in num\_str:

return float(num\_str)

else:

return int(num\_str)

# Test the get\_number() function

value = get\_number()

print(value)

def get\_direction(prompt = "Enter the required direction: "):

while True:

# Remove any leading or trailing whitespace

direction\_str = input(prompt).strip()

# Make sure that only a single valid character is accepted

# in either upper or lower case

if all(c in 'NESWnesw' for c in direction\_str):

if len(direction\_str) == 1:

break

print('Invalid input - try again!')

return direction\_str.upper() # Return only the upper case character

# Test the get\_direction() function

direction = get\_direction()

print(direction)

sudo i2c detect –y 0

sudo i2c detect –y 1

i2cdetect –y 0

i2cdetect –y 1

i2cset –y 0 0x62 0x0f 0xff

i2cset –y 0 0x62 0x00 0x00

i2cset –y 0 0x62 0x04 0xd0

import subprocess

return\_code = subprocess.call(["i2cset –y 0 0x62 0x09 0x00"], shell = True)

print(return\_code)

# Import the required libraries

import time

import subprocess

# Initial voltage = 0V

voltage = 0

step = 0

while step < 10:

voltage = step \* 0.3

# Calculate the digital code

volts = voltage \* 4096 / 3.3

# Convert the digital code to two bytes

code = int(volts)

first = int(code / 255)

second = code - (first \* 256)

# Format the two hexadecimal values

value1 = format(first, '#04x')

value2 = format(second, '#04x')

# Build the command string

command\_string = "i2cset -y 0 0x62 " + value1 + " " + value2

# Make the sub-process call

return\_code = subprocess.call([command\_string], shell=True)

# Hold the voltage steady for two seconds

time.sleep(2)

# Increment the step counter

step = step + 1

# After exiting the loop set the output back to 0V

command\_string = "i2cset -y 0 0x62 0x00 0x00"

return\_code = subprocess.call([command\_string], shell=True)

# MCP4725 DAC for Python 3.x

import subprocess

voltage = 0

while True:

value = input("Output voltage: ")

voltage = float(value)

while voltage >= 0 and voltage < 3.3:

volts = voltage \* 4096 / 3.3

code = int(volts)

first = int(code / 255)

second = code - (first \* 256)

value1 = format(first, '#04x')

value2 = format(second, '#04x')

command\_string = "i2cset -y 0 0x62 " + value1 + " " + value2

return\_code = subprocess.call([command\_string], shell=True)

if return\_code == 0:

print("Success - output now set to ", value + " V")

else:

print("Error setting DAC - output not changed!")

voltage = -1

# MCP4725 DAC with amplifier for Python 3.x

import subprocess

voltage = 0

while True:

value = input("Output voltage: ")

voltage = float(value)

while voltage >= 0 and voltage < 10.1:

volts = voltage \* 4096 / 11

code = int(volts)

first = int(code / 255)

second = code - (first \* 256)

value1 = format(first, '#04x')

value2 = format(second, '#04x')

command\_string = "i2cset -y 0 0x62 " + value1 + " " + value2

return\_code = subprocess.call([command\_string], shell=True)

if return\_code == 0:

print("Success - output now set to ", value + " V")

else:

print("Error setting DAC - output not changed!")

voltage = -1

# MCP4725 DAC for Python 2.x

# Import the required libraries

import time

import smbus

# Set the bus to use (for Rev.2 change 0 to 1)

bus = smbus.SMBus(0)

# DAC's I2C bus address

DAC = 0x62

# Initial voltage = 0V

voltage = 0

step = 0

while step < 11:

voltage = step \* 0.3

# Calculate the digital code

volts = voltage \* 4096 / 3.3

# Convert the digital code to two bytes

code = int(volts)

first = int(code / 255)

second = code - (first \* 256)

bus.write\_word\_data(DAC,first,second)

# Hold the voltage steady for two seconds

time.sleep(2)

# Increment the step counter

step = step + 1

# After exiting the loop set the output back to 0V

bus.write\_word\_data(DAC,0x00,0x00)

# MCP4725 DAC for Python 2.x

# Import the required libraries

import time

import smbus

# Set the bus to use (for Rev.2 change 0 to 1)

bus = smbus.SMBus(0)

# DAC's I2C bus address

DAC = 0x62

voltage = 0

while True:

value = input("Output voltage: ")

voltage = float(value)

while voltage >= 0 and voltage < 3.3:

volts = voltage \* 4096 / 3.3

code = int(volts)

first = int(code / 255)

second = code - (first \* 256)

bus.write\_word\_data(DAC,first,second)

voltage = -1

# MCP4725 DAC with amplifier for Python 2.x

# Import the required libraries

import time

import smbus

# Set the bus to use (for Rev.2 change 0 to 1)

bus = smbus.SMBus(0)

# DAC's I2C bus address

DAC = 0x62

voltage = 0

while True:

value = input("Output voltage: ")

voltage = float(value)

while voltage >= 0 and voltage < 10.1:

volts = voltage \* 4096 / 11

code = int(volts)

first = int(code / 255)

second = code - (first \* 256)

bus.write\_word\_data(DAC,first,second)

voltage = -1