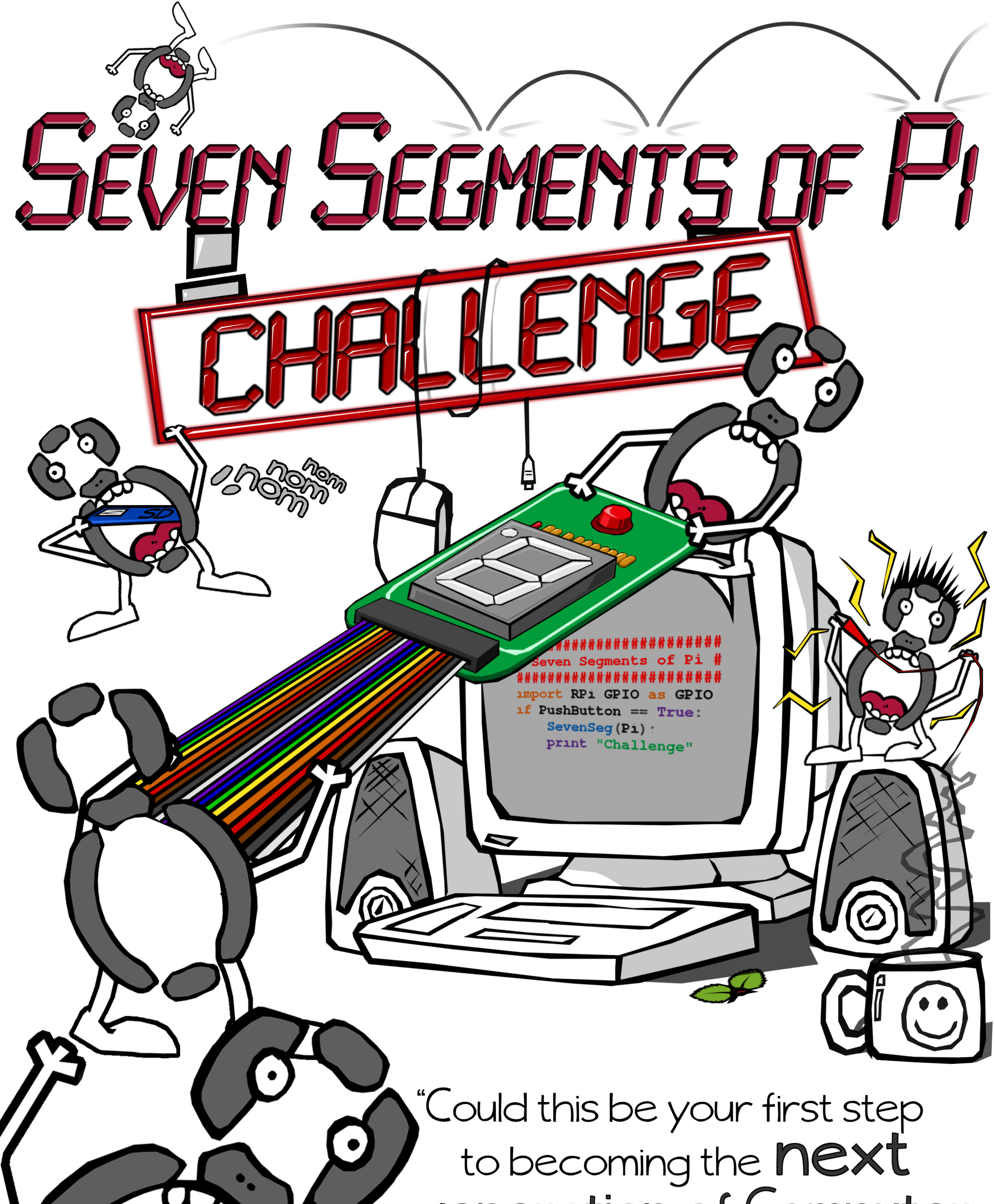
# Seven Segments of Pi Challenge



```
#######################
# Seven Segments of Pi #
#######################
import RPi.GPIO as GPIO
if PushButton == True:
    SevenSeg(Pi):
    print "Challenge"
```

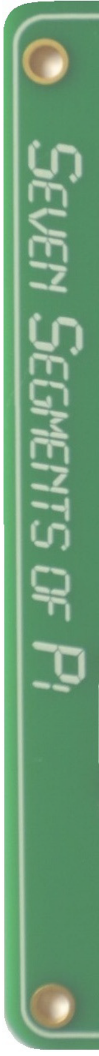
"Could this be your first step to becoming the **next** generation of Computer Games Designer?"

An *innovations in education* Product

## Contents

## Seven Segments of Pi Challenge

*You may have never written any software before!*
*You may not even know what software is!!*
*So this "Seven Segments of Pi" Challenge aims*
*not only to show you what software is,*
*but, if you complete the Challenge,*
*you will have written the software for your own computer game!*
*And who knows…*
*this could be your first step*
*to becoming the next generation of Computer Games Designer!!!*



*To complete the Seven Segments of Pi Challenge you will need to…*

***Assemble the Seven Segments of Pi PCB Kit***

***Write the software for a PiDice***

***Write the software for a PiStopWatch***

*You will then have the software skills needed to…*

***Write the software for you first Computer Game…***

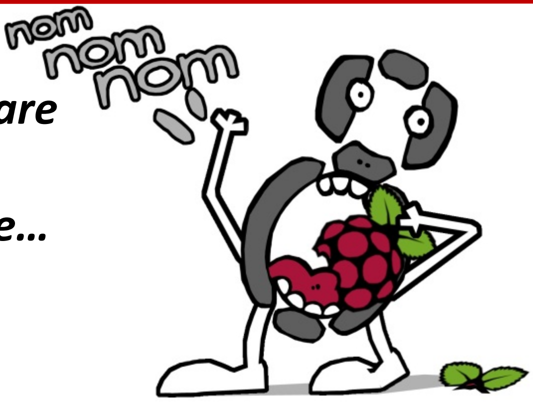
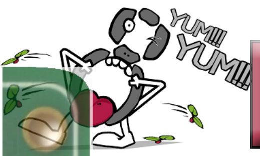


***Figure Eight my Pi***



**What is the "*Seven Segments of Pi*"?**
It is an electronic circuit which attaches to your **Raspberry Pi** Computer to create what may be the ***Simplest Games Console in the World***!

…however although it may be simple, it includes all of the elements of any games console – '***Control***' and '***Display***'. A single Red PushButton acts as your 'Control' and a Seven Segment LED acts as your 'Display' - but unlike other games consoles, this one is designed to run the software written by you!

**So, how does it do it?**
The **Raspberry Pi** Computer has a number of "General Purpose Input/Output" or "GPIO" signals which can be controlled by software. A GPIO signal configured as an input can, for instance, allow your software to sense if a PushButton has been pressed. A GPIO signal configured as an output can, for instance allow your software to illuminate a Light Emitting Diode (LED). These GPIO Signals are available on the **Raspberry Pi**'s **P1** Connector. So by attaching your own electronics to the **P1** Connector you can control it from your software!

The ***Seven Segments of Pi*** is controlled via 8 of these GPIO signals. One is used as an input from the Red PushButton and seven are used as outputs to illuminate the segments of the Seven Segments Display. Often a Seven Segment Display is just used to display a single digit number however, since your software has control of each individual segment, it doesn't *have* to be used to just for numbers, as you will see!

**What is the *Seven Segments of Pi Challenge*?**
The challenge is to build your ***Seven Segments of Pi*** Games Console and write the software to run on it with software written in the programming language "**Python**".

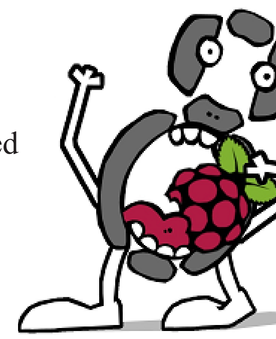
The ***Seven Segments of Pi*** is supplied as a kit of components, so your first task is to solder the components to the Printed Circuit Board (PCB) by following the **Seven Segments of Pi Assembly Instructions**. For this you will need a soldering iron, solder, a pair of wire cutters and a small pair of pliers.

Once your ***Seven Segments of Pi*** is assembled, plug it into your **Raspberry Pi** and plug in the "***SSPi***" SD Card (if included with the kit) or download from www.SevenSegmentsOfPi.com.

The SD Card image not only has the normal **Raspberry Pi** "New-Out-Of-the-Box-Software" (NOOBS) ***Raspbian*** Operating System but is also preloaded with all of the software necessary to get you started programming in **Python** with the minimum of effort!

After a few exercises to get you started with **Python**, your first main software task is to write a program for an electronic dice, a **PiDice**! Starting with an initial **Python** program provided on the SD Card, you need to modify the software in seven steps until you have a working **PiDice** and in doing so you will learn how to detect if the PushButton is pressed, how to illuminate the segments of the display and how to generate random numbers in software.

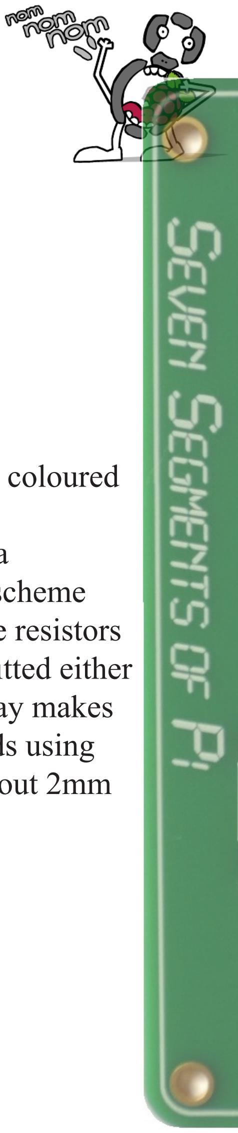
Your second software task is to write a program for an electronic stopwatch, a **PiStopWatch**! Once again starting with an initial **Python** program provided on the SD Card, you need to modify the software in seven steps until you have a working **PiStopWatch** and in doing so you will learn how to write "State Machines" and how to generate software controlled sound effects!



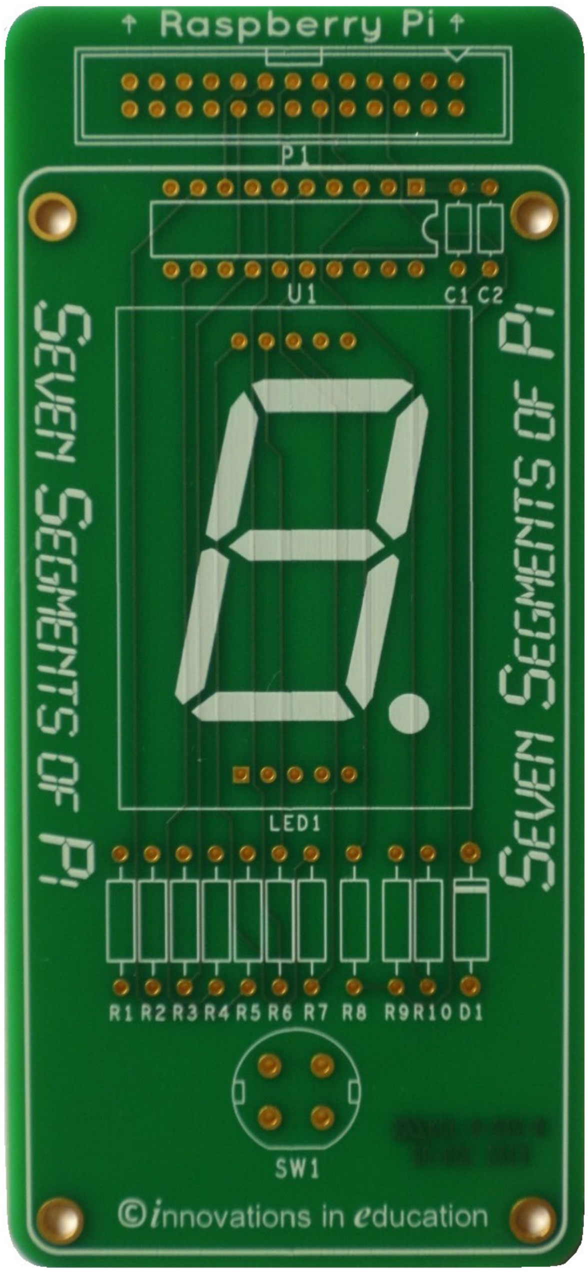
You will now have the software skills needed to write your first computer games program! Once again, starting with an initial **Python** program provided on the SD Card, you need to modify the software in seven steps, using the software skills learnt from writing the **PiDice** and **PiStopWatch** programs until you have written the software for the game ***Figure Eight my Pi***!

If you make a working ***Figure Eight my Pi*** game you will have completed the ***Seven Segments of Pi Challenge*** …but don't stop there! Take a look at the ***"Figure Eight My Pi Extras"!*** Eight ideas to make your game even better…or maybe come up with your own ideas to make the game better! And look at the description of the **"*Seven Segments of Pi*"** electronics. If you understand the electronics you could try designing your own electronics to connect to the **Raspberry Pi**!

# Seven Segments of Pi Assembly Instructions



This is the ***Seven Segments of Pi*** Printed Circuit Board (PCB).

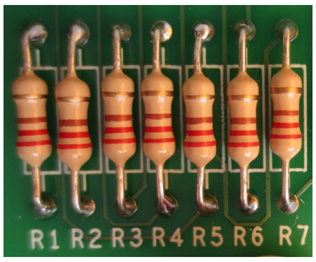


All components are fitted on this top surface and soldered on the underside. To assemble the kit you will need the following:

- ***Soldering Iron***
- ***Solder***
- ***Wire Cutters***
- ***Small Pliers***

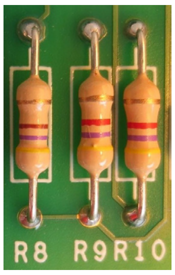
To ease assembly, fit and solder the smallest components first, followed by the next tallest and so on. The Resistors and Capacitors can be fitted either way round but **`P1,U1,LED1,D1`** ***and*** **`SW1`** ***MUST BE FITTED AS SHOWN !!!***

**Fit and Solder `R1-7`**



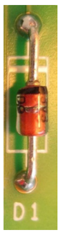
**`R1-7`** are 220 ohm Resistors marked with coloured bands RED, RED, BROWN, GOLD.
(See "Understanding the Electronics" for a description of the Resistor colour coding scheme and an explanation of the function of these resistors and other components). Resistors can be fitted either way round but fitting them all the same way makes it look neater! Once soldered crop the leads using your Wire Cutters so that no more than about 2mm protrudes on the underside.

**Fit and Solder `R8-10`**



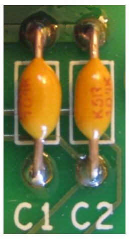
**`R8`** is a 470 ohm Resistors marked with coloured bands YELLOW, PURPLE, BROWN, GOLD.
**`R9`** & **`R10`** are 4k7 ohm Resistors marked with coloured bands YELLOW, PURPLE, RED, GOLD.

**Fit and Solder `D1`**

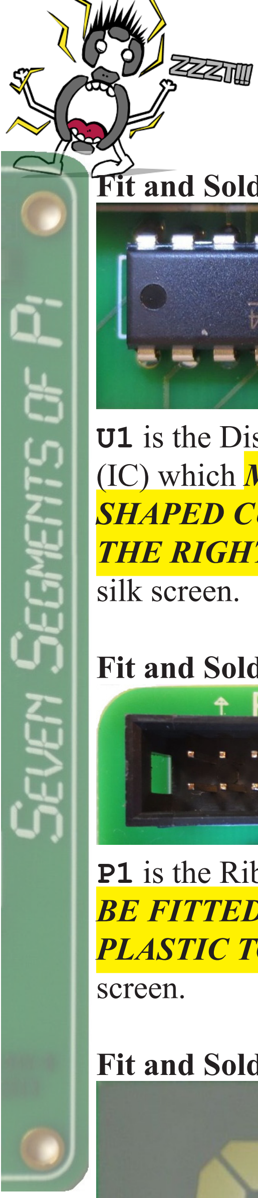


**`D1`** is a Zener Diode which ***MUST BE FITTED WITH ITS BLACK BAND AT THE TOP*** as indicated by the white band on the silk screen

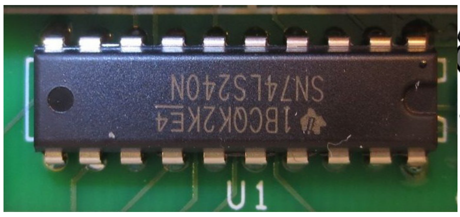
**Fit and Solder `C1-2`**



**`C1`** & **`C2`** are 100nF Capacitors and can be fitted either way round

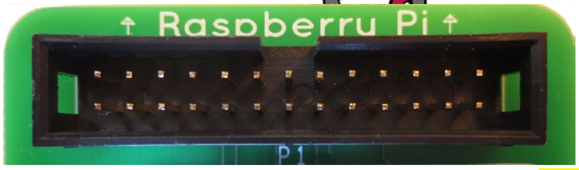


**Fit and Solder `U1`**



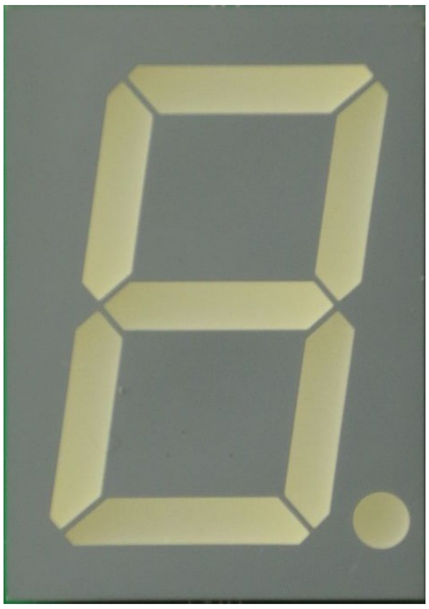
`U1` is the Display Driver Buffer Integrated Circuit (IC) which ***MUST BE FITTED WITH THE 'U' SHAPED CUT-OUT IN ITS PLASTIC LID TO THE RIGHT*** as indicated by the 'U' shape on the silk screen.

**Fit and Solder `P1`**



`P1` is the Ribbon Cable Connector which ***MUST BE FITTED WITH THE CUT-OUT IN THE PLASTIC TO THE TOP*** as indicated by the silk screen.

**Fit and Solder `LED1`**



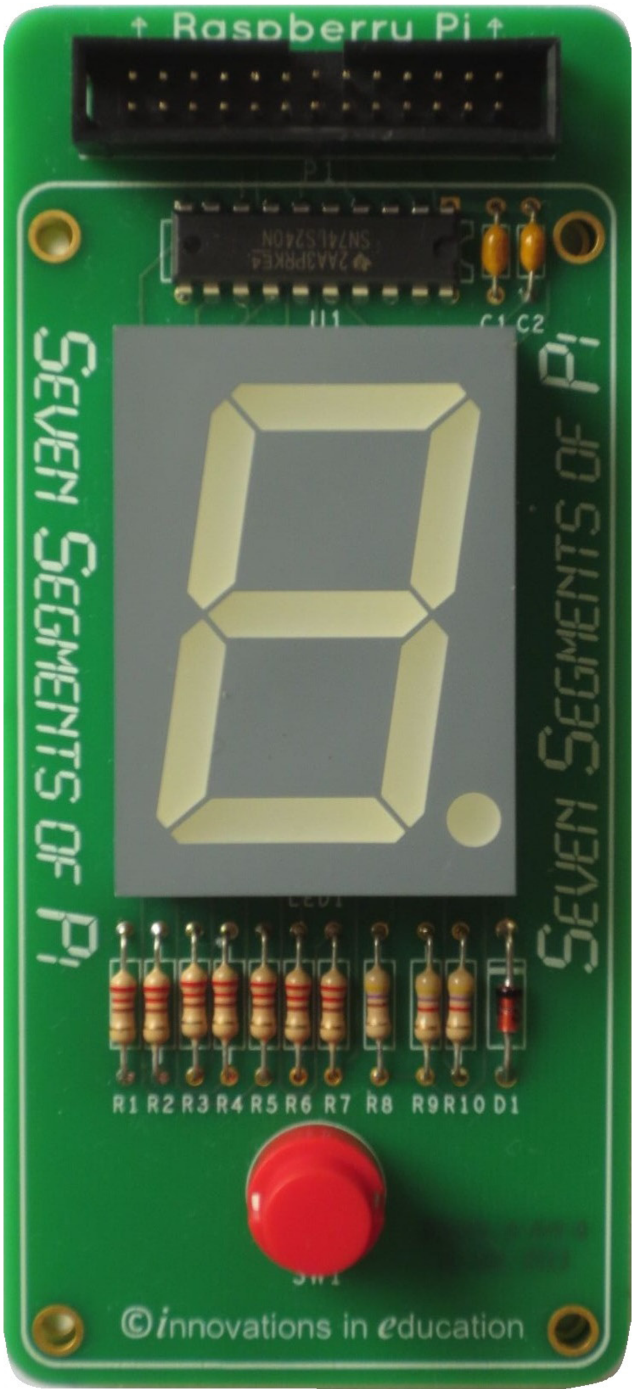
`LED1` is the Seven Segments Display which ***MUST BE FITTED WITH THE DECIMAL POINT TO THE BOTTOM RIGHT*** as indicated by the silk screen.

**Fit and Solder `SW1`**

SW1 is the Red PushButton which ***MUST BE FITTED WITH THE FLAT PLASTIC SIDE TO THE BOTTOM*** as indicated by the silk screen.

**Fit the four Self Adhesive Feet**
These fit on the underside of the PCB in the positions marked by four circles on the silk screen.

The fully assembled ***Seven Segments of Pi*** PCB looks like this.



## Testing and Troubleshooting

Before attaching your ***Seven Segments of Pi*** to the Raspberry Pi check the following:-

**Checks:-**
C1) Are **P1,U1,D1,LED1** and **SW1** fitted the correct way round?
C2) Are all pins soldered?
C3) Are there any solder shorts between pins?
C4) Are any Resistor, Capacitor or Diode wires touching adjacent components?

If everything looks OK attach one end of the Ribbon Cable to Connector **P1** of the ***Seven Segments of Pi*** and the other end to Connector **P1** of your Raspberry Pi.

The ***Seven Segments of Pi*** connector is shrouded so the ribbon cable will only fit in it one way round. The Raspberry Pi connector **P1** is unshrouded so take care aligning the Ribbon Cable connector with the Raspberry Pi connector pins.

Plug the "***SSPi***" SD Card into your Raspberry Pi and power it up. The first time you power it up you will need to install the "Raspbian Operating System with Seven Segments of Pi files" by selecting:-

+SSPi **Install (i)**

The ***Seven Segments of Pi*** should power up with ***all seven segments illuminated*** and when you press the PushButton the Decimal Point on the display should illuminate too!

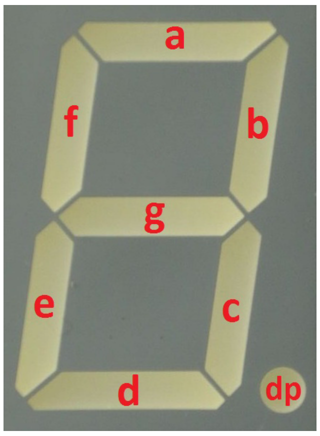
If so, you are ready to run the "**Python**" test program as described on the next page! If not, follow the troubleshooting tips below to find the assembly problem:-

**Troubleshooting Tips:-**
T1) Firstly re-check points C1, C2, C3 & C4 listed above.

T2) If your Raspberry Pi fails to boot it could be that your Power Supply cannot supply the extra current of about 80mA needed by the Seven Segments of Pi. It is recommended that you use a Power Supply that can supply 5V at a current of 1A (1000mA) or more.

T3) If nothing illuminates there is probably a problem with the power connection so check the Ribbon Cable is plugged in correctly at both ends and check the soldering of **P1** Connector pins 2 & 6, **U1** pins 10 & 20 and LED1 pins 1 & 5.

T4) If some segments illuminate but others do not there is probably a soldering problem associated with the faulty segment(s). The seven segments are referred to by the letters **a** to **g** with segment **a** being at the top then going clockwise around the display they are **b**, **c**, **d**, **e** & **f** with **g** being the middle segment as shown below (**dp** is the decimal point).



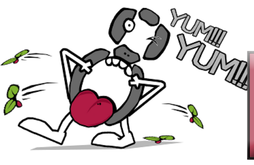
If segment **a** does not illuminate check the soldering of **P1** pin 11, **U1** pins 2 & 18, **R1** and **LED1** pin 7 as shown below.

For other segments look at the Circuit Diagram in the section "**Understanding the Electronics**" to find the pin numbers to check.

T5) If the Decimal Point doesn't illuminate when the PushButton is pressed check the soldering of **SW1,C2,U1** pin 17, **R8** and **LED1** pin 8.

T6) If the Decimal Point *does* illuminate when the PushButton is pressed, but when you run the **Python** Test Program nothing happens when you press the PushButton, check the soldering of **U1** pin 3, **R10**, **D1** and **P1** pin 7.

For any other problems read the section on "**Understanding the Electronics**". This might help you to work out which other connection(s) may be causing the fault.



## Running Python Test Program

To test that your ***Seven Segments of Pi*** board is fully working run the **Python** Test Program "**Seven_ Segments_OneTwoThree.py**" included on the "***SSPi***" SD Card. Follow the instructions below and if everything is working the Seven Segments display should show the numbers 1, 2, 3 after the PushButton is pressed.

When the **Raspberry Pi** powers up wait until it 'Boots' to the prompt below and type in the text highlighted in **bold**

*If you don't see anything on your screen there may be a problem with the video configuration for your monitor. The "SSPi" SD Card is based on the latest* **Raspberry Pi** *"New-Out-Of-the-Box-Software"(NOOBS). Do a Google search for "***Raspberry Pi** *NOOBS Video Troubleshooting" for help.*

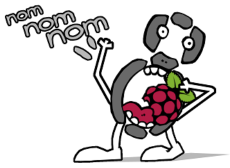
raspberrypi login: **pi**
Password: **raspberry**

pi@raspberrypi ~$ **sudo startx**
Starts up Linux IDE (Windows like GUI interface)

**Start** > **Programming** > **IDLE** (Opens Python Shell)
**File** > **Open…** > **/home/pi/share/PiDice/Seven_Segments_OneTwoThree.py**
Opens IDLE editor and shows software for above Python program
**Run** > **Run Module** (or just press "**F5**")
*Python Shell* window opens and runs Python Program
Prompt "`Press PushButton to Start`" is displayed

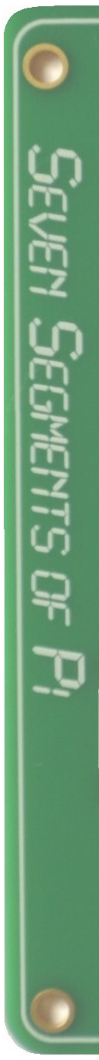
**Press PushButton** on Seven Segments of Pi
Seven Segment Display should display '1' '2' '3' with approx 1 second intervals
and 'one' 'two' 'three' should be displayed at the same time on the *Python Shell* window
**Press PushButton** again and the above should repeat
**File** > **Close >** Kill? **> OK** to stop program and close *Python Shell* window

# Getting Started with Python



Look at the ***"Seven_Segments_OneTwoThree.py"*** program!
It is written in the Computer Language …

python™



If you are new to **Python** ***don't*** try and understand it! Just try making these simple changes highlighted in `yellow` and see what they do! As you make these changes hopefully you will ***start*** to understand what they are doing! If you have written **Python** software before you may understand what they are doing from the start!

***Tip*** – make your changes look similar to the existing software, in particular if the existing lines you are copying are indented then indent your new lines. You will find out later that this indentation is an essential part of the Python language!

**Print Your Name**
Change the line

```
        print "One"
```

to

```
        print "Your Name"
```

Select **File** > **Save**
and run it again
**Run** > **Run Module** (or just press "**F5**")
(If you haven't saved the file it will prompt you to do so when you select Run Module)
***Do you see where your name is printed?***

***Explanation!*** Whenever a Python program encounters a `print` command it prints the text inside the quotation marks on the *Python Shell* window, so you can now get it to print any text you want!

**Change the Numbers Displayed**
Change the lines

```
def one(): # Define function 'one' which makes GPIO b,c = True to display number '1'
    GPIO.output(11, False) # a
```

to

```
def one(): # Define function 'one' which makes GPIO b,c = True to display number '1'
    GPIO.output(11, True) # a
```

and run it again
***Can you see what has changed?***

***Explanation!*** Setting `GPIO.output 11` to `True` illuminates segment 'a', the top segment on the Seven Segment Display. When the program now tries to display the number `one` it illuminates segment 'a' in addition to segments b & c, so `one` now appears as the number 7!

**Change the Speed of the Counter**
Change the line

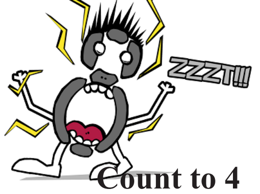
```
delay = 1
```

to

```
delay = 2
```

and run it again
***Has it changed speed?***

***Explanation!*** The constant `delay` is used in `time.sleep(delay)` which is the delay (in seconds) between displaying each number. By changing it to 2 you have slowed it down. Try changing it to `delay = .5` to speed it up!



## Count to 4

In the space just above `def SevenSeg(x):` add

```
def four():# Define function 'four' which makes GPIO?,?,?,? = True to display number '4'
    GPIO.output(11, ????)  # a
    GPIO.output(12, ????)  # b
    GPIO.output(13, ????)  # c
    GPIO.output(15, ????)  # d
    GPIO.output(16, ????)  # e
    GPIO.output(18, ????)  # f
    GPIO.output(22, ????)  # g
```

The quickest way to do this is to copy the whole of `defthree()` and paste it above `defSevenSeg(x)` then modify the parts highlighted in `yellow` above. You will also have to work out which `????` need to be `True` and which need to be `False` to illuminate the correct segments to display the number 4.

***Tip*** - *A quick way to Copy and Paste is to use the mouse to select the text you wish to copy, then press Ctrl-C to copy it, then Ctrl-V to paste it.*

…and run it again

***Does it do anything different from before? No?***

***Explanation!*** The program now knows ***how*** to display the number 4 but has not yet been ***instructed*** to display the number 4, so it still just displays 1,2,3!

So add the lines highlighted in `yellow` below

```
def SevenSeg(x):
    if x == 1:
        one()
    elif x == 2:
        two()
    elif x == 3:
        three()
    elif x == 4:
        four()
```

…and run it again

***Does it do anything different from before? No?***

***Explanation!*** The program now knows how to ***ask*** to display the number 4 but still has not yet been ***instructed*** to display the number 4, so once again it still just displays 1,2,3!

So add the lines highlighted in `yellow` below

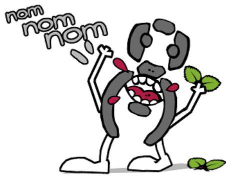
```
        print "Three"
        SevenSeg(3)
        time.sleep(delay)
        print "Four"
        SevenSeg(4)
        time.sleep(delay)
```

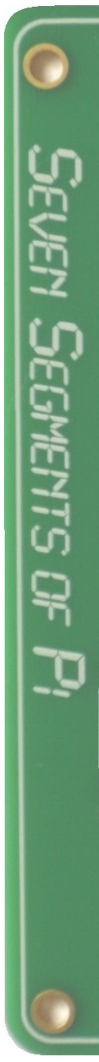
…and run it again

***Does it now count to 4?***

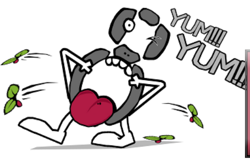
***Explanation!*** The program now has the ***instruction*** to display the number 4 using the line `SevenSeg(4)` When it gets to this line it jumps to `def SevenSeg(x):` with the value of `x == 4.` Since `x` equals `4` it jumps to `def four():` and illuminates the segments to display the number 4!!!

*If you have any problems refer to the section "**Understanding Python**" to help give you a better understanding of the Python language, "**Python Troubleshooting**" for help understanding error messages and **Python Debug Control**" for debugging your software!*

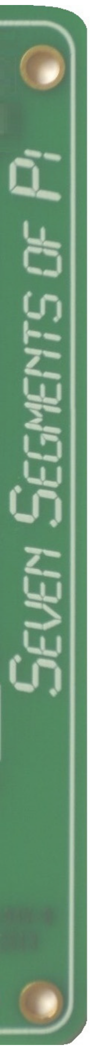




```
##################################################################
# Seven Segments of Pi - Seven_Segments_OneTwoThree.py           #
##################################################################
# Description:-                                                  #
# When PushButton is pressed                                     #
# GPIOs drive Seven Segment Display with numbers 1, 2, 3         #
# with 1 second delay between numbers as a simple Counter        #
##################################################################
#!/usr/bin/env python    #allows program to be run from command line
import time              #time package allows programmable delays in the software
import RPi.GPIO as GPIO  #RPi.GPIO package allows control of GPIO by software
GPIO.setmode(GPIO.BOARD) #RPi.GPIO package numbers GPIO by their Raspberry Pi Connector pin number
GPIO.setwarnings(False)  #Disables GPIO Warning Messages

GPIO.setup(7, GPIO.IN)   #GPIO 7 is input from Push Button Switch
                                                                        _______
GPIO.setup(11, GPIO.OUT) #GPIO 11 output illuminates Segment a         |   a   |
GPIO.setup(12, GPIO.OUT) #GPIO 12 output illuminates Segment b        f|       |b
GPIO.setup(13, GPIO.OUT) #GPIO 13 output illuminates Segment c         |_______|
GPIO.setup(15, GPIO.OUT) #GPIO 15 output illuminates Segment d         |   g   |
GPIO.setup(16, GPIO.OUT) #GPIO 16 output illuminates Segment e        e|       |c
GPIO.setup(18, GPIO.OUT) #GPIO 18 output illuminates Segment f         |_______|
GPIO.setup(22, GPIO.OUT) #GPIO 22 output illuminates Segment g             d

def one(): # Define function 'one' which makes GPIO b,c = True to display number '1'
    GPIO.output(11, False) # a
    GPIO.output(12, True)  # b
    GPIO.output(13, True)  # c
    GPIO.output(15, False) # d
    GPIO.output(16, False) # e
    GPIO.output(18, False) # f
    GPIO.output(22, False) # g

def two(): # Define function 'two' which makes GPIO a,b,d,e,g = True to display number '2'
    GPIO.output(11, True)  # a
    GPIO.output(12, True)  # b
    GPIO.output(13, False) # c
    GPIO.output(15, True)  # d
    GPIO.output(16, True)  # e
    GPIO.output(18, False) # f
    GPIO.output(22, True)  # g

def three(): # Define function 'three' which makes GPIO a,b,c,d,g = True to display number '3'
    GPIO.output(11, True)  # a
    GPIO.output(12, True)  # b
    GPIO.output(13, True)  # c
    GPIO.output(15, True)  # d
    GPIO.output(16, False) # e
    GPIO.output(18, False) # f
    GPIO.output(22, True)  # g

def SevenSeg(x): # Define function 'SevenSeg' which calls function 'one' 'two' or 'three' depending on 'x'
    if x == 1:
        one()
    elif x == 2:
        two()
    elif x == 3:
        three()

poll = .1   # Define Constant for PushButton 'poll' delay
delay = 1   # Define Constant for Counter delay

# Start of Counter 1, 2, 3 Program Execution
print "Press PushButton to Start"    # Printed on Raspberry Pi *Python Shell* Window
while True:                          # while True: means run this loop forever
    PushButton = GPIO.input(7)       # Check PushButton input
    if PushButton == False:          # if PushButton has not been pressed
        time.sleep(poll)             # wait 0.1 of a second before checking (polling) again
    else:                            # else PushButton has been pressed, so
        print "One"                  # Print "One" on Raspberry Pi *Python Shell* Window
        SevenSeg(1)                  # Call function 'SevenSeg' with a value of x=1
        time.sleep(delay)            # wait a second
        print "Two"                  # Print "Two" on Raspberry Pi *Python Shell* Window
        SevenSeg(2)                  # etc
        time.sleep(delay)
        print "Three"
        SevenSeg(3)
        time.sleep(delay)
```



# Understanding Python



The section **"Getting Started with Python"** is probably all you need to know to get started writing your **PiDice** software so you could skip this bit! But by reading this section you will have a better understanding of the **Python** language, in particular you will understand the why some words are `coloured` and why some lines are **indented**.

In **Python**...

**Comments** - Comments are automatically highlighted in `red`. For instance:-

```
        print "Your Name"  # This text is coloured red so it is a comment
```

The hash character **#** indicates the rest of the line contains comments and the IDLE Editor will automatically colour them `red`. Comments are not part of the program but they help you to understand what the program is doing. ***Read the comments in the programs to understand what they are doing!***

**Keywords** – are automatically highlighted in `orange`. These have a special meaning, for instance

`import` – indicates the name of an imported package e.g. time & RPi.GPIO
`def` – indicates the definition of a function
`if,elif,else & while` controls the flow of the program

**Print Command**

`print` – prints text e.g. `"Your Name"` to *Python Shell* Window

**Functions** – Often a program will need to do the same thing many times so rather than repeating the lines again and again they can be written as a 'function'. The keyword `def` indicates a function definition. The function must be given a name which is highlighted in `blue` and the contents of the function are the indented lines which follow. In a program whenever the name of the function is 'called' the program jumps to these lines, executes them, then jumps back again. The test program ***"Seven_Segments_OneTwoThree.py"*** uses four functions:- `one,two,three` and `SevenSeg`

**Functions** `one,two,three` – define which GPIO are `True` or `False` to display the numbers 1,2 and 3.

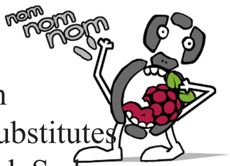
**Function** `SevenSeg` – calls one of the functions above depending on the value of '**x**'. When calling this function a value for '**x**' must be given – see below

**Indentation** – indicates parts of program which are part of a `def` function definition, or are part of an `if`, `elif`, `else` statement, or are part of a `while` statement. ***This is important to get your program to run!!*** Look at this example taken from the ***"Seven_Segments_OneTwoThree.py"*** program:-

```
def SevenSeg(x):
    if x == 1:   # This line is indented so is part of function SevenSeg
        one()    # This line is double indented so is only executed if x == 1:
    elif x == 2: # This line is indented so is still part of function SevenSeg
        two()    # This line is double indented so is only executed if x == 2:
    elif x == 3: # This line is indented so is still part of function SevenSeg
        three()  # This line is double indented so is only executed if x == 3:
poll = .1        # This line is NOT indented so is NOT part of function SevenSeg
```

**Boolean Constants** - `True`, `False` – are automatically highlighted in `purple`. They can be used to set GPIO output pins. For instance in these programs `True` turns a segment 'on' and `False` turns it 'off'. It is also used to test if a GPIO input is 'on' (`True`) or 'off' (`False`)

**GPIO** - These programs uses the Raspberry Pi's GPIO (General Purpose Input/Output) signals. These signals allow the Raspberry Pi to control and respond to other electronic circuits attached to the GPIO Connector. The GPIO Connector is attached to the ***Seven Segments of Pi*** electronics via the ribbon cable. Seven GPIO signals are used as outputs to control the LED display. When a GPIO output is set to `False`, the LED segments it controls will be 'off' but when it is set to `True`, the LED segment will be 'on'
One GPIO signal is used as an input from the PushButton. If this GPIO input is 'read' and it is `False`, the PushButton is ***not*** being pressed but if it is `True`, the Push Button ***is*** being pressed.
To make use of the GPIO the program must `import` the package **`GPIO.RPi`**.



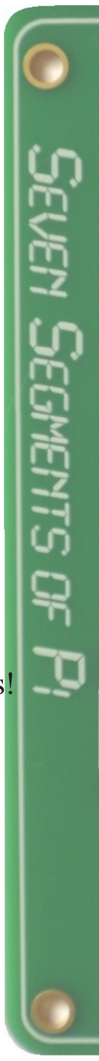
**User Defined Constants** - `poll, delay` - defines 2 user defined constants used by test program ***"Seven_Segments_One_Two_Three.py"*** – Whenever the program encounters these constants it substitutes their value, so for instance `delay = 1` is used by `time.sleep(delay)` to sleep for a second. So by changing it to `delay = 2` it will sleep for 2 seconds.
To use `time.sleep` the program must `import` the package `time`.

**Start of Program Execution** – In ***"Seven_Segments_OneTwoThree.py"*** the first line of the program to 'execute' is after the comment `# Start of Counter` (everything before this line is just setting things up ready for the program to use!)

`while True:` means everything indented below it loops forever.

The program then 'reads' the GPIO input from the PushButton Switch.
If it is `False` (i.e. ***not*** being pressed) the program 'sleeps' for a period of time set by the constant '`poll`' after which it returns to the line where it 'reads' the GPIO input from the PushButton Switch.
If it is `True` (i.e. it ***is*** being pressed) the program prints "`One`", calls function `SevenSeg` giving '`x`' the value of '`1`'. It then 'sleeps' for a period of time set by the constant '`delay`'. It then repeats this for "`Two`" and "`Three`". Once complete it returns to the line where it 'reads' the GPIO input from the PushButton Switch.

To learn more about the **Python** Programming Language go to www.python.org and www.pygame.org or read the book "**Programming the Raspberry Pi – Getting Started with Python**" by Simon Monk.



## Python Troubleshooting

When you run your programs you might get Error Messages or the program might run but not do what you wanted it to do! ***This is to be expected!*** However you must correct it to make your program work. Here are a few common errors, but if you get any others, do a Web Search and hopefully you will find some useful hints!

There are 4 common types of error:-

- `Invalid Syntax` – ***The location of the error will be highlighted in red***
- `Indented Block` – ***The location of the error will be highlighted in red***
- `Name Error` – ***The line number where the error is located will be in the error message***
- `RunTime Error` – ***The line number where the error is located will be in the error message***

**Invalid Syntax**
Most Syntax Errors will report
"`There is an error in your program: Invalid Syntax`"
This normally means there is something wrong just ***before*** the red bar.
Common Syntax Errors:- `if`, `elif`, `else` & `while` with single equals sign or missing colon
For example:-

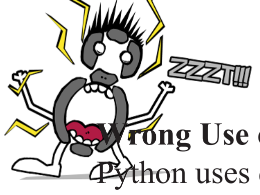
```
if PushButton = False: # Syntax Error! Only has a single equals, needs a double equals
if PushButton == False # Syntax Error! Missing :
if PushButton == False:# Correct!
```

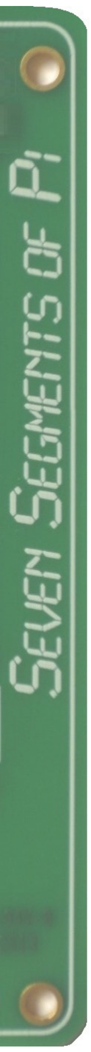
…but when assigning a value to a constant or a variable it must be a single equals sign and no colon

```
delay == 1      #Name Error! Assigning a value to a constant needs a single equals
delay = 1:      #Syntax Error! Assigning a value to a constant doesn't needs a :
delay = 1       #Correct!
```

**Wrong Case**
Python is case sensitive, so for example P is different from p, hence `print` is a keyword but **Print** is not!
This mostly results in a Name Error message such as
`NameError: name 'false' is not defined` and gives the line number where the error occurs.
Some examples of using the Wrong Case:-

```
if PushButton == false: # Name Error! Wrong Case! False should have an upper case F
if Pushbutton == False: # Name Error! Wrong Case! PushButton defined with upper case B
If PushButton == False: # Syntax Error! Wrong Case! if must use lower case i
if PushButton == False: # Correct!
Sevenseg(1) # Name Error! Wrong Case! The function SevenSeg defined with upper case S
SevenSeg(1) # Correct!
```





**Wrong Use of Quotation marks**

Python uses quotation marks to distinguish between text strings and variables
For example

```
print "x" # This is correct if you want to print the letter x
          #   but wrong if you want to print the value of x
print x   # This is correct if you want to print the value of x
          #   but wrong if you want to print the letter x
```

**Incorrect Use of Indentation**

Python uses indentation to structure the program. These indentations are part of the language so must be correct.

**Indented Block Error**

Many Indentation Errors will report
"`There is an error in your program: expected an indented block`"
and will highlight the location in red. The location of the error will normally be just ***before*** the red bar.

Examples of Indented Block Errors:-

```
def one():
GPIO.output(11, False)      # Indented block error. All lines of the function
                            #   after the def must be indented
def one():
    GPIO.output(11, False) #Correct indentation

if x == 1:
one()          # Indented block error. All lines which are part
               #   of the if statement must be indented
if x == 1:
    one()      # Correct indentation
```

***Incorrect indentation may however not give you an Error Message, instead it may mean the program doesn't do what you want it to do!*** For instance try changing the indentation in the test program to be as shown below. If you run it you will find it doesn't report an error but it has changed how the program behaves!

```
if PushButton == False:
    time.sleep(poll)
else:
    print "One"
    SevenSeg(1)
    time.sleep(delay)
    print "Two"
    SevenSeg(2)
    time.sleep(delay)
print "Three"          #These lines are not indented like the other lines
SevenSeg(3)            #so are no longer part of the "if else" statement
time.sleep(delay)      #hence are executed when PushButton is both True and False
```

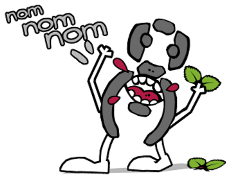
**Other Indentation Error Messages**

You may get an indentation error caused by having indentation using a mixture of <Tabs> and <Spaces>.
This can be rectified by **Edit**>**SelectAll**>**Format**>**Untabify Region**>Columns per tab? **> 4**
This will change all <Tabs> in your program to be 4 <Spaces>.

**RunTime Error Message**

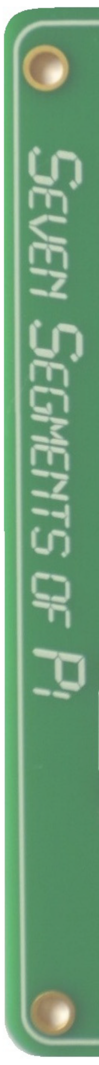
If you get a `RunTime Error` which says `Try running as root!`
The most likely reason is that when you started up, you typed
pi@raspberrypi ~$  **startx**
where you should have typed
pi@raspberrypi ~$  **sudo startx**

The '**sudo**' before the **startx** means you are running in 'Super User' mode (also called 'Supervisor', 'Administrator' or 'Root' mode). You need to be in this mode to control the GPIO pins.



**IDLE or IDLE3**
The ***Raspbian*** has 2 Python Programming editors, IDLE and IDLE3.
All software for the Seven Segments of Pi Challenge is designed to run using IDLE, *not* IDLE3
If, for instance, you run the test program using IDLE 3 you will get an **Invalid Syntax** error at the line
`print "Press PushButton to Start"`
This is the *correct* syntax for IDLE which uses inverted commas for print statements but the *incorrect* syntax for IDLE3 which uses brackets. So the correct syntax for IDLE3 would be
`print (Press PushButton to Start)`

## Python Debugger



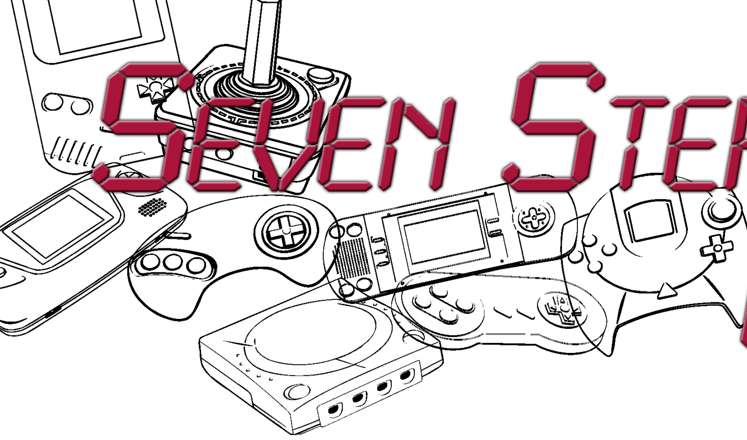
If, when you run your program, it runs without errors but doesn't do what you want it to do, you must debug it! To help you, the ***Raspberry Pi*** IDLE Editor has a ***Debugger*** which allows you to run your program a one step at a time so you can check what it is doing, or run it until it reaches a specific line known as a "***Breakpoint***" where it will stop so you can check what it has done. Here is a brief introduction to the ***Debugger*** and an example of how to use it:-

To enable the ***Debugger***:-

**Start** > **Programming** > **IDLE** (Opens Python Shell)
**Debug**>**Debugger**
The Debug Control window opens.

The Debug Control window gives 5 options (these are greyed out until you run the program)
 <**Go**> Runs the Program and Stops when it reaches a Breakpoint.
 <**Step**> Runs the next line of the Program and Stops, known as Single Stepping.
 <**Over**> Similar to <**Step**> but treats a Function call as a Single Step.
 <**Out**> In a Function it runs remainder of the Function and stops at the line when the program returns.
 <**Quit**> Quits the Debugger.
The Debug Control window also displays the current value of variables used by your program.

To see how it can be used try this:-
**File** > **Open…** > **/home/pi/share/PiDice/Seven_Segments_OneTwoThree.py**
**<Right-Click>** on the line `SevenSeg(1)` **<Set Breakpoint>** and Line 71 highlights in `yellow`
You have now set a ***Breakpoint*** so the program will stop (break) when it gets to this line.
**Run** > **Run Module** (or "**F5**")
 Select <**Go**> in the Debug Control Window (Debug Controls go grey indicating the program is running)
Prompt "`Press PushButton to Start`" is displayed.
**Press PushButton** on Seven Segments of Pi.
"`One`" is displayed and the Debug Controls go black indicating the program has 'hit' the ***Breakpoint*** and
`>'__main__'.<module>(), line 71: SevenSeg(1)` is displayed indicating it has stopped on line 71 and at the bottom of the Debug Control window `PushButton 1` shows PushButton = 1 (i.e. True).
 Select <**Step**> and the program stops at the line
`>'__main__'.<module>(), line 53: if x==1:` indicating the program has jumped to the first line of the Function `SevenSeg(x)` and at the bottom of the Debug Control window `x 1` shows variable x=1.
 Select <**Step**> again and the program stops at the line
`>'__main__'.<module>(), line 54: one()` indicating the program has jumped to the second line of the Function `SevenSeg(x)`.
 Select <**Step**> again and the program stops at the line
`>'__main__'.<module>(), line 26: GPIO.output(11, False)` indicating the program has jumped to the first line of the Function `One()`.
 Select <**Out**> and the program runs the remaining lines of Function `One()` and stops at the line
`>'__main__'.<module>(), line 72: time.sleep(delay)` i.e. the next line after the Function.
 Select <**Over**> and the program runs the Function `time.sleep(delay)` and stops at the line
`>'__main__'.<module>(), line 73: print "Two"` i.e. the next line after the time delay.



# Seven Steps to Pi Dice

*So, now that you understand the "Seven_Segments_OneTwoThree.py" program, it is time for you to write some software!*
*Your first challenge is to write the software for an electronic dice…*
*…a "PiDice"!!*

***Your starting point…***

**Start** > **Programming** > **IDLE** (Opens Python Shell)
**File** > **Open…** > **/home/pi/share/PiDice/PiDice_Step0.py**
This has identical code to ***"Seven_Segments_OneTwoThree.py"*** and is the starting point for your PiDice
***Tip*** - *Before you start, save it with a different file name so that you can always go back to PiDice_Step0.py*
**File** > **SaveAs…** > **/home/pi/share/PiDice/PiDice_Step1.py**
*then before you start each new step SaveAs a new file name so that if you get in a muddle you can always go back to your last working software!!!*
***Tip*** - *In case your SD Card is damaged or lost take occasional **backups** as described in the section "**Taking Backups and Transferring Files onto your** Raspberry Pi"*

***And this is the software you need to write…***

***D1***) **Add a new function '`def four()`' for number 4 with GPIO for Segments b,c,f,g = `True` and a,d,e = `False`.** ***Tip*** *– Copy and paste using Ctrl-C and Ctrl-V as described in "Getting Started with Python".*
*Run it! Does it do anything different from before?*

***D2***) **Change Counter to count from 1 to 4 after PushButton pressed**
*Run it! Now it should do something different from before!*

***D3***) **Add new functions '`def five()`' for number 5, and '`def six()`' for number 6**
*Run it! Does it now count to 6?*

***D4***) **Change Counter to count from 1 to 6 after PushButton pressed**
*Run it! Now it should count to 6!*

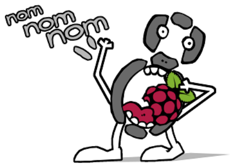
***D5***) **Add new function '`def blank()`' with all segments 'off' and change it to count *down* 5,4,3,2,1,blank after PushButton pressed**
*Run it! Now it should count down from 5 to 1 and then go blank*

***D6***) **Change it to display a random number from 1-6 after PushButton pressed**
***Tip*** - *look at "Seven_Segments_Random.py" for how to generate random numbers in Python*
*Run it! This is the difficult step. It may take you some time to get it to work!*

***D7***) **Change it to count *down* 5,4,3,2,1,blank after PushButton pressed *then* display the random number from 1-6**
*Run it! And if it works...*

***…you now have an Electronic PiDice!!!***

# Seven_Segments_Random.py



In Step ***D6*** you need to generate random numbers. The **Python** program ‘Seven_Segments_Random.py’ below gives you an example of how to generate random numbers in **Python**. The lines highlighted in **yellow** are the important ones.

```
############################################################################
# Seven Segments of Pi - Seven_Segments_Random.py Random Number Generation #
############################################################################
# Description:-                                                            #
# When PushButton is pressed                                               #
# a Random Number between 1 - 6 is printed on the *Python Shell* Window    #
# Example software for copying into PiDice program, etc                    #
############################################################################
#!/usr/bin/env python    #allows program to be run from command line
import random            #random package allows random number generation in the software
import time              #time package allows programmable delays in the software
import RPi.GPIO as GPIO  #RPi.GPIO package allows control of GPIO by software
GPIO.setmode(GPIO.BOARD) #Sets RPi.GPIO package to number GPIO by their Raspberry Pi Connector pin number

GPIO.setup(7, GPIO.IN)   #GPIO 7 is input from Push Button Switch

poll = .1  # Define Constant for PushButton 'poll' rate

# Start of Random Number Generator Program Execution
print "Press PushButton to Start"
while True :
    PushButton = GPIO.input(7)           # Check PushButton input
    if PushButton == False:              # if PushButton has not been pressed
        time.sleep(poll)                 # wait 0.1 of a second before checking again
    else:                                # else PushButton has been pressed, so
        x = random.randrange(1,7)        # get a random number between 1 & 6
        print x                          # print this random number to Raspberry Pi *Python Shell* Window
```

Run the program as follows:-

**Start** > **Programming** > **IDLE** (Opens Python Shell)
**File** > **Open…** > **/home/pi/share/PiDice/Seven_Segments_Random.py**
Opens IDLE editor and shows software for above Python program
**Run** > **Run Module** (or just press “**F5**”)
*Python Shell* window opens and runs Python Program
Prompt “`Press PushButton to Start`” is displayed

**Press PushButton** on Seven Segments of Pi
Whilst the PushButton is pressed a series of random numbers between 1 and 6 is printed on the *Python Shell* window

Note that **`random.randrange`** generates a random number of any value from the first number in brackets to ***one less*** than the second number!

***Knowing how to generate random numbers in Python will be needed when you come to write the software for your game!***

# State Machines in Python

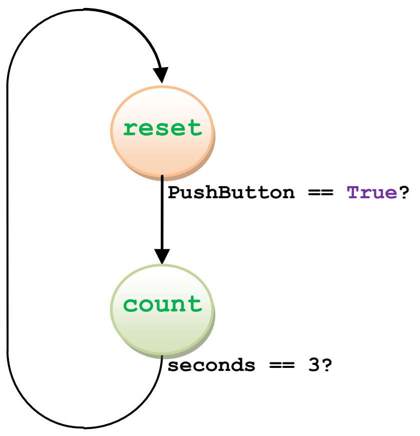
Many computer programs include "State Machines" to control the flow of the program and you will need to know how to create them when you come to write the software for your next task, the **PiStopWatch**. State Machines can be created in **Python** by a series of "`if`" statements where each "`if`" statement defines what is done when in that "`state`". Opposite is a program called 'Seven_Segments_OneTwoThree_StateMachine.py'. Follow the instructions below to run it and see what it does.

With the Raspberry Pi powered up, logged in and IDE (Windows like GUI interface) running…

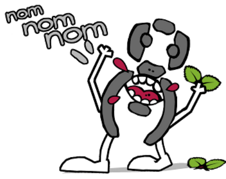
**Start** > **Programming** > **IDLE** (Opens Python Shell)
**File** > **Open…** > **/home/pi/share/PiStopWatch/Seven_Segments_OneTwoThree_StateMachine.py**
Opens IDLE editor and shows software for above Python program
**Run** > **Run Module** (or just press "**F5**")
*Python Shell* window opens and runs Python Program
Prompt "`Press PushButton to Start`" is displayed

**Press PushButton** on Seven Segments of Pi and it appears to do exactly the same as the program "***Seven_Segments_One_Two_Three.py***" used as the starting point for the **PiDice** program, that is, every time the PushButton is pressed the numbers 1, 2, 3 appear at approximately 1 second intervals on the Seven Segments display, but this software is written using a "State Machine".

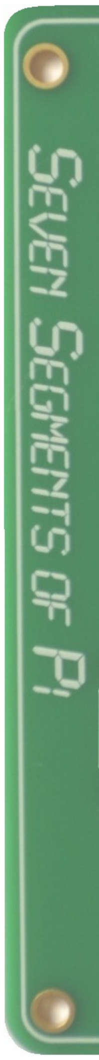
When designing State Machine it is often useful to show the flow of the State Machine as a diagram, so below is a diagram showing the flow of the State Machine in "***Seven_Segments_OneTwoThree_StateMachine.py***". Each state is given a name. This State Machine has just two states called `"reset"` and `"count"`. The arrows show that when PushButton is `True` (i.e. the PushButton has been pressed) the State Machine flows from state `"reset"` to state `"count"` (however while PushButton is `False` it stays in state `"reset"`). Similarly it stays in state `"count"` until the count of seconds reaches 3 at which point it moves from state `"count"` back to state `"reset"`.

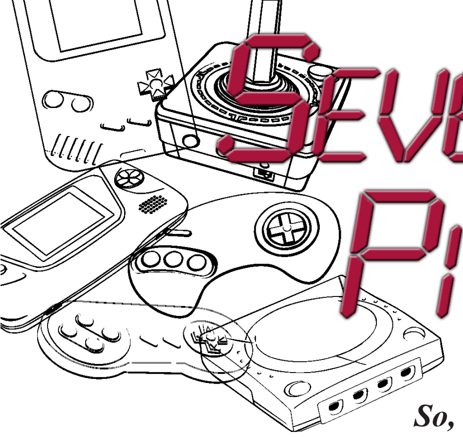


Now look at the lines opposite highlighted in `yellow`. These are the lines of software needed to make this State Machine. The State Machine has 2 states called `"reset"` and `"count"` which must always be in inverted commas. The State Machine starts in state `"reset"`. When in state `"reset"` the seconds and tenths counters are set to 0 and the PushButton's GPIO input is 'read'. If the PushButton has been pressed (`True`) the State Machine moves to state `"count"` otherwise it remains in state `"reset"`, sleeps for a 0.1 second delay and 'reads' the PushButton again. When in state `"count"` it counts up displaying seconds 1,2,3 and until `seconds == 3:` when it jumps back to state `"reset"`.

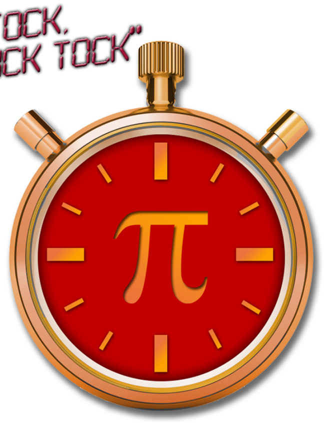


```
#####################################################################
# Seven Segments of Pi - Seven_Segments_OneTwoThree_StateMachine.py #
#####################################################################
# When PushButton is pressed                                        #
# GPIOs drive Seven Segment Display with numbers 1, 2, 3            #
# with 1 second delay between numbers as a simple Counter           #
# implemented as a State Machine                                    #
#####################################################################
#!/usr/bin/env python    #allows program to be run from command line
import time              #time package allows programmable delays in the software
import RPi.GPIO as GPIO  #RPi.GPIO package allows control of GPIO by software
GPIO.setmode(GPIO.BOARD) #RPi.GPIO package numbers GPIO by their Raspberry Pi Connector pin number
GPIO.setwarnings(False)  #Disables GPIO Warning Messages

GPIO.setup(7, GPIO.IN)   #GPIO 7 is input from Push Button Switch
                                                                        _______
GPIO.setup(11, GPIO.OUT) #GPIO 11 output illuminates Segment a         |   a   |
GPIO.setup(12, GPIO.OUT) #GPIO 12 output illuminates Segment b        f|       |b
GPIO.setup(13, GPIO.OUT) #GPIO 13 output illuminates Segment c         |_______|
GPIO.setup(15, GPIO.OUT) #GPIO 15 output illuminates Segment d         |   g   |
GPIO.setup(16, GPIO.OUT) #GPIO 16 output illuminates Segment e        e|       |c
GPIO.setup(18, GPIO.OUT) #GPIO 18 output illuminates Segment f         |_______|
GPIO.setup(22, GPIO.OUT) #GPIO 22 output illuminates Segment g             d

def one(): # Define function 'one' which makes GPIO b,c = True to display number '1'
    GPIO.output(11, False) # a
    GPIO.output(12, True)  # b
    GPIO.output(13, True)  # c
    GPIO.output(15, False) # d
    GPIO.output(16, False) # e
    GPIO.output(18, False) # f
    GPIO.output(22, False) # g

def two(): # Define function 'two' which makes GPIO a,b,d,e,g = True to display number '2'
    GPIO.output(11, True)  # a
    GPIO.output(12, True)  # b
    GPIO.output(13, False) # c
    GPIO.output(15, True)  # d
    GPIO.output(16, True)  # e
    GPIO.output(18, False) # f
    GPIO.output(22, True)  # g

def three(): # Define function 'three' which makes GPIO a,b,c,d,g = True to display number '3'
    GPIO.output(11, True)  # a
    GPIO.output(12, True)  # b
    GPIO.output(13, True)  # c
    GPIO.output(15, True)  # d
    GPIO.output(16, False) # e
    GPIO.output(18, False) # f
    GPIO.output(22, True)  # g

def SevenSeg(x): # Define function 'SevenSeg' which calls function 'one' 'two' or 'three' depending on 'x'
    if x == 1:
        one()
    elif x == 2:
        two()
    elif x == 3:
        three()

delay = 1   # Define Constant for delay

# Start of Counter 1, 2, 3 Program Execution
state = "reset"                              # Start State Machine in state "reset"
print "Press PushButton to Start"            # Printed on Raspberry Pi *Python Shell* Window
while True:                                  # while True: means run this loop forever
    if state == "reset":                     ####### state "reset" ######
        seconds = 0                          # zero 'seconds' counter
        tenths = 0                           # and Zero tenths
        PushButton = GPIO.input(7)           # Check PushButton input
        if PushButton == True:               # if PushButton has been pressed...
            state = "count"                  # ...move to state "count"
        else:
            time.sleep(delay)                # if not, wait for 0.1 sec before checking PushButton again

    if state == "count":                     ####### state "count" ######
        time.sleep(delay)                    # wait for 0.1 second
        tenths = tenths +1                   # increment tenths counter
        if tenths == 10:                     # when tenths gets to 10...
            seconds = seconds + 1            # ...increment seconds counter
            tenths = 0                       # and reset the tenths counter back to 0
            SevenSeg(seconds)                # and display the seconds count on Seven Segment Display
            print "%d.%d seconds"%(seconds,tenths) # print the seconds and tenths on *Python Shell* Window
        if seconds == 3:                     # if seconds count has reached 3...
            state = "reset"                  # ...move to state "reset"
```





# Seven Steps to Pi StopWatch



***So, now that you have written the software for a "PiDice"
it is time for you to write some more software!
Your second challenge is to write the software for an electronic stopwatch…
…a "PiStopWatch"!!***

***Your starting point…***

**Start** > **Programming** > **IDLE** (Opens Python Shell)
**File** > **Open…** > **/home/pi/share/PiStopWatch/PiStopWatch_Step0.py**
This is the same as ***"Seven_Segments_OneTwoThree_StateMachine.py"*** and is the starting point for your PiStopWatch.
***Tip -** As before, start each new step save as a new file name…*
**File** > **SaveAs…** > **/home/pi/share/PiStopWatch/PiStopWatch_Step1.py**

***And this is the software you need to write…***

***S1*) Add functions '`def four()`' '`def five()`' '`def six()`' '`def seven()`' '`def eight()`' '`def nine()`' '`def zero()`' for numbers 4,5,6,7,8,9,0 with the GPIO for Segments `True` and `False` to display these numbers.**
*Run it! Does it do anything different from before?*

***S2*) Change Counter to start from 0 and count to 9 after the PushButton pressed.**
*Run it! Does it now count from 0 to 9?*

***S3*) Add a sound effect!**
**Generate sound `"phaser.wav"` when moving from State `"reset"` to `"count"`.**
(Look at "***Seven_Segments_Sound.py***" for how to generate sounds in Python)
*Run it! Do you get any sound?*

***S4*) Change Counter to move to State `"stop"` when the PushButton pressed a 2nd time.**
*Don't forget that checking the PushButton needs the* two *lines.*

```
PushButton = GPIO.input(7)          # Check PushButton input
if PushButton == True:              # if PushButton has been pressed...
```

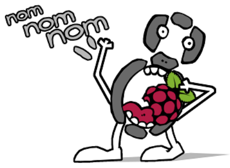
*Run it! Does it now start to count on 1st PushButton press and stop on 2nd PushButton press?
If not, why not?(See **PiStopWatch State Machine** to understand why not!)*

***S5*) Change Counter so that it uses 2 States:- State `"count0"` counts seconds and moves to `"count1"` when the PushButton is *released* (`False`). State `"count1"` continues counting seconds and moves to state `"stop0"` when the PushButton is *pressed* (`True`).**
*Run it! Does it now start to count on 1st PushButton press and stop on 2nd PushButton press?*

***S6*) Add all 6 States shown in the *PiStopWatch State Machine* diagram, changing from state to state depending on whether PushButton is *pressed* (`True`) or *released* (`False`) as shown. In states `"reset0"` and `"reset1"` it should display '0'. In states `"count0"` and `"count1"` it should display the seconds count and in states `"stop0"` and `"stop1"` it should display the final count.**
*Run it! Does it now behave like a Stop Watch?*

***S7*) Add more sound effects!**
**Generate sound `"tick.wav"` every time the seconds count is incremented.**
**Generate sound `"twang.wav"` when moving from State `"count1"` to `"stop0"`.**
**Generate sound `"cackle.wav"` when moving from State `"stop1"` to `"reset0"`.**

***…you now have an Electronic PiStopWatch!!!***



# Seven_Segments_Sound.py

In Step ***S3*** and ***S7*** you need to add the software to generate sound effects. The **Python** program "***Seven_Segments_Sound.py***" below gives you an example of how to generate sound effects in **Python**. The lines highlighted in **yellow** are the important ones. If you are using a VGA or DVI monitor, connect speakers (or headphones) to the 3.5mm Analog Audio Output Connector of the **Raspberry Pi** to play the sound effects. If you are using an HDMI TV, the sound effects will be played via the TV. If you don't hear the sound effects, look at the section "**Troubleshooting Sound**"

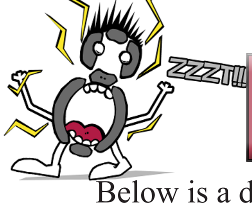
```
####################################################################
# Seven Segments of Pi - Seven_Segments_Sound.py                   #
####################################################################
# Description:-                                                    #
# Press PushButton to Generate a Test Sound                        #
####################################################################

#!/usr/bin/env python        # allows program to be run from command line

import pygame                # pygame package allow sounds to be generated
from pygame.locals import *

import time                 #time package allows programmable delays in the software
import RPi.GPIO as GPIO     #RPi.GPIO package allows control of GPIO by software
GPIO.setmode(GPIO.BOARD)    #RPi.GPIO package numbers GPIO by their Raspberry Pi Connector pin number
GPIO.setwarnings(False)     #Disables GPIO Warning Messages

GPIO.setup(7, GPIO.IN)      #GPIO 7 is input from Push Button Switch

global TEST                                  # Defines the names of a sound called 'TEST'
pygame.mixer.pre_init(44100,-16,2,512)       # Sets Sound parameters (freq,size,channels,buffer)
pygame.init()                                # Initialise pygame package
                                             # Load the sound files:-
TEST = pygame.mixer.Sound("doh.wav")         # selects the sound file to be used for 'TEST'

poll = .1                                    # Define Constant for PushButton 'poll' delay
delay = 1                                    # Define Constant for delay

# Start of Sound Test Program Execution
print "Press PushButton to Start"            # Printed on Raspberry Pi *Python Shell* Window
while True:                                  # while True: means run this loop forever
    PushButton = GPIO.input(7)               # Check PushButton input
    if PushButton == True:                   # if PushButton has been pressed...
        print "Playing Test Sound!"          # Printed on Raspberry Pi *Python Shell* Window
        sound = TEST                         # and play the sound TEST
        sound.play()
        time.sleep(delay)                    # then wait for 1 second before checking PushButton again
    else:
        time.sleep(poll)                     # if not, wait for 0.1 sec before checking PushButton again
```

Run the program as follows:-

**Start** > **Programming** > **IDLE** (Opens Python Shell)
**File** > **Open…** > **/home/pi/share/PiStopWatch/Seven_Segments_Sound.py**
Opens IDLE editor and shows software for above Python program
**Run** > **Run Module** (or just press "**F5**")
*Python Shell* window opens and runs Python Program
Prompt "`Press PushButton to Start`" is displayed

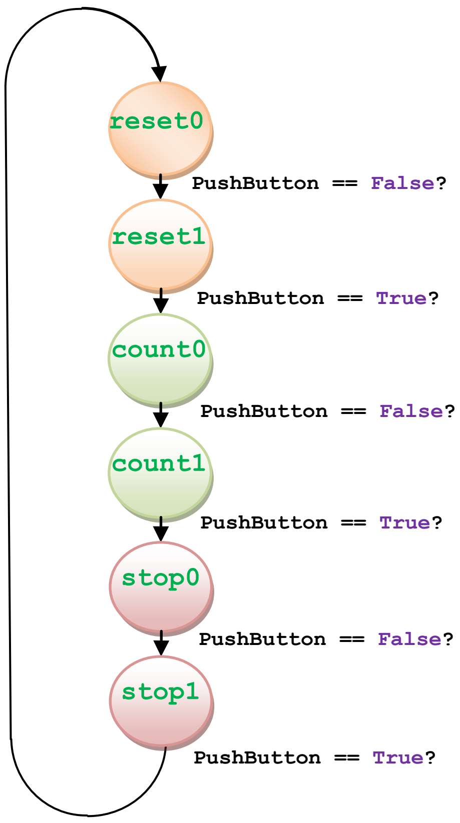
**Press PushButton** on the Seven Segments of Pi
Whilst the PushButton is pressed you should hear the sound effect!
Any sound effect available as a `.wav` or `.ogg` file can be played so long as the file is in the same directory as your **Python** program. Edit the above program changing `"doh.wav"` to `"badswap.wav"` and run the program again.

***Knowing how to generate sound effects in Python will be needed when you come to write the software for your game!***

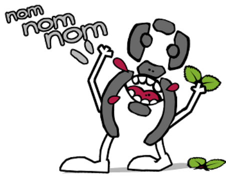


## PiStopWatch State Machine

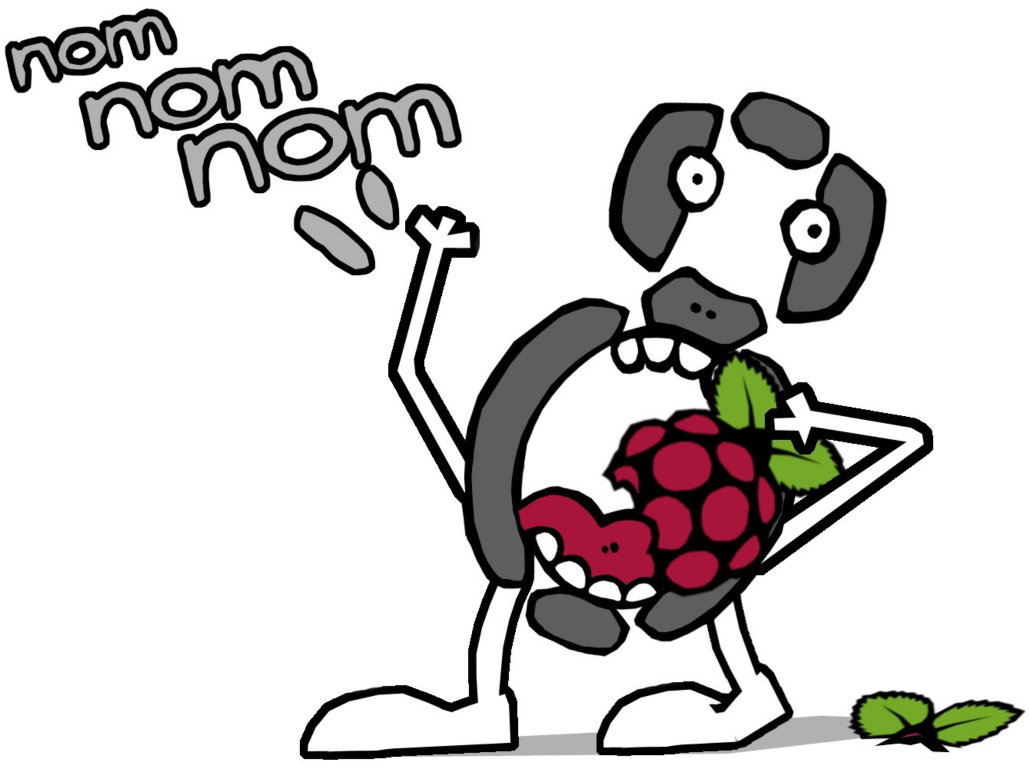
Below is a diagram showing the flow of the State Machine needed for the **PiStopWatch**. On the face of it a Stop Watch requires 3 states. `"reset"`, where it displays zero until the PushButton is pressed, `"count"` where it displays the seconds as they count up until the PushButton is pressed a second time and `"stop"` where the final seconds count is displayed. With a third PushButton press to return to state `"reset"`. But, since the software can 'read' the PushButton many times a second, if the software were written with just these 3 states it would move from state `"reset"` to state `"count"` as the PushButton was pressed but then immediately move to state `"stop"` then state `"reset"` and so on until the PushButton is released. What is required after moving to state `"count"` is to detect that the PushButton has been *released* before detecting it has been *pressed* again to move to state `"stop"`. This can be achieved by having 6 states which alternately wait until the PushButton is `False` then `True` before moving to the next state as shown.



***YOU ARE NOW READY TO WRITE THE SOFTWARE FOR YOUR FIRST COMPUTER GAME !!!***



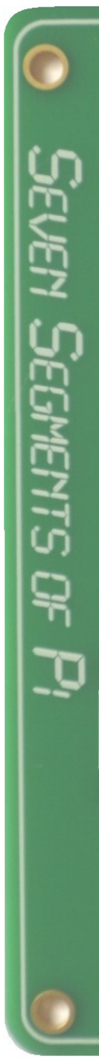
The Marketing Department of ***Seven Segments of Pi Enterprises*** has created a computer games character called the ***PiSeg*** (you may have seen him, *or is it her*, before!)



…and devised a game to run on the ***Seven Segments of Pi*** Games Console called…

# FIGURE EIGHT MY PI

…and ***you*** have been commissioned to write the software!



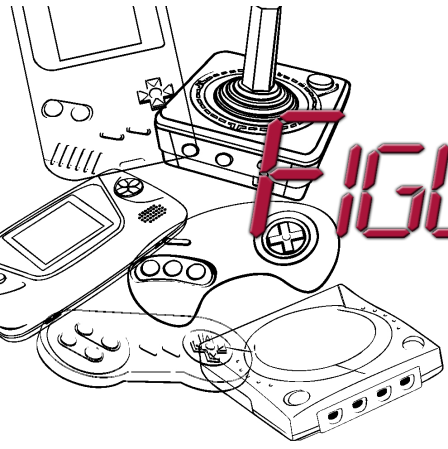
Your brief is to create a game that is:-

- ***Challenging***
- ***Interesting, and***
- ***Fun to Play***

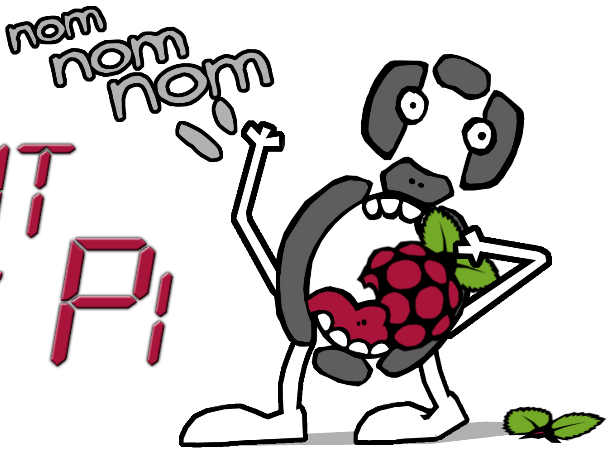
So start by making a working game. Then adjust the game's speed making it challenging to play. Try out different sound effects making it fun to play. And add features making it interesting to play. Experiment with different ideas and at each stage try it out on friends and family and use their feedback to guide you.

As with other computer games organisations ***Seven Segments of Pi Enterprises*** requires that you keep you software confidential, **so do not share your Python software with others or post it on the internet** but feel free to demonstrate your game in action to anyone and everyone!

***TURN OVER TO DISCOVER THE RULES OF THE GAME Figure Eight My Pi!***



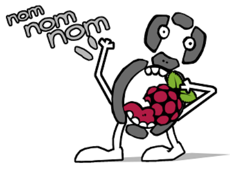
# Figure Eight my Pi



The cunning "PiSeg" is munching its way around your Pi...
...but when it chomps up and down the sides watch out!
There is a chance it will try and "Figure Eight Your Pi"
by attempting to gobble the prized middle segment!
If it tries to, you must shoot it!!!
If you are on target you will score a point...
...but if you let it munch past, you will lose a point!
And don't waste your ammunition!
If you shoot when the cunning "PiSeg"
is not munching the prized middle segment...
...you will lose another point!

How many can <u>you</u> score?

# Seven Steps to Figure Eight My Pi



***So, this is how to play "Figure Eight My Pi"***
***all you need to do is write the software!***

***Your starting point…***

**Start** > **Programming** > **IDLE** (Opens Python Shell)
**File** > **Open…** > **/home/pi/share/FigureEightMyPi/FigureEightMyPi_Step0.py**
This is the starting software for your game. Run it to see what it does.
It makes a single segment (the "*PiSeg*") appear to step around the display in a clockwise "Figure Zero"

**Look at the software.**
It is written as a '**State Machine**' with 6 states (See **Figure Eight My Pi State Machine Step 0** Diagram)
State `"a"` turns segment '**a**' '*on*' for 0.5 seconds. It then moves to state `"b"` which turns segment '**b**' '*on*' for 0.5 second, then **c**, **d**, **e** and **f** then back to state `"a"`.

***Tip*** *- Before you start each step save it with a different file name so that you could always go back a step*
**File** > **SaveAs…** > **/home/pi/share/FigureEightMyPi/FigureEightMyPi_Step1.py**
*and for each new step save as a new file name so that can always go back to your last working software!!!*

***And this is the software you need to write…***

***F1*****) Change the state machine so that the *PiSeg* appears to step around the display in a "Figure Eight" and reduce the delay to 0.3 seconds to speed it up! (See Figure Eight My Pi State Machine Step 1** Diagram**)**
*Run it! Does it work?*

***F2*****) Add sound effect called '`STEP`' at the end of each state just before it changes to the next state.
Look at Seven_Segments_Sound.py from the PiStopWatch Workshop for how to add sound effects.
You can use any sound effect in the directory /home/pi/share/FigureEightMyPi.**
*Run it! You should now hear the sound the **PiSeg** makes as it munches round your Pi!*

***F3*****) Change the state machine so that the *PiSeg* when it goes up or down the side has a 50% chance of carrying on in a "Figure Zero" and a 50% chance of going across the middle in a "Figure Eight"
Looks at Seven_Segments_Random.py from the PiDice Workshop for how to generate random numbers.
*Hint* – you will need more states since, for instance when segment 'a' is '*on*' it now might be going clockwise or anticlockwise! These need to be 2 different states! (See Figure Eight My Pi State Machine Step 3** Diagram**)**
*Run it! You should now see the **PiSeg** sneak across the middle …but not every time!*

***F4*****) Read the PushButton at the end of each state** (just before it plays the sound effect) **and print "True" or "False" to the Terminal depending on whether it is being pressed.**
*Run it! Press the PushButton and check what is printed!*
***Hint* – once working '*comment out*' all prints. You no longer need them and they will slow the game down!**

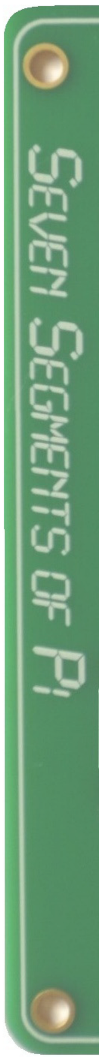
***F5*****) Add a score. Start the score at 0. Add 1 to the score if PushButton is '`True`' at the end of all 'g' segment states. Subtract 1 if PushButton is '`False`' at the end of all 'g' segment states. And subtract 1 if PushButton is '`True`' at the end of all other states. Print the score whenever it changes.**
*Run it! You now almost have a working game!*

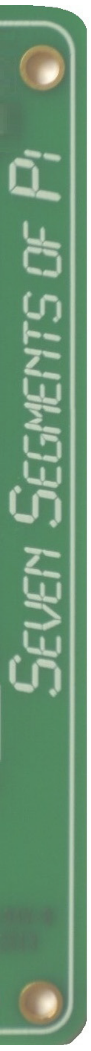
***F6*****) Add a sound effects called '`HIT`' whenever you add 1 to the score. Add a sound effects called '`MISS`' whenever you subtract 1 from the score.**
*Run it! You should now be able to tell if you have had a 'hit' or a 'miss'!*

***F7*****) Keep a count of how many times the *PiSeg* goes through any 'g' segment state. Also reduce the '`delay`' by 0.01 seconds at the end of every 'g' segment state so the *PiSeg* gradually speeds up!
End the game when `count = 10`.**
*Run it! And if it works...*

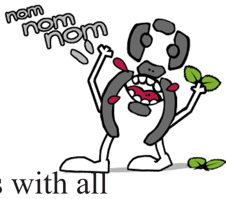
***…you have now written the game***
***"Figure Eight my Pi"!!!***

***…but can you make it even better?***



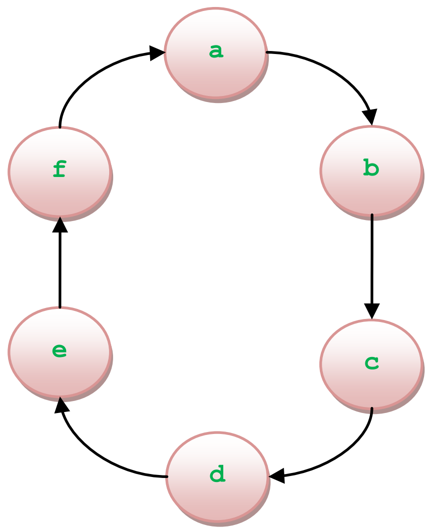


```
######################################################################
# Seven Segments of Pi - FigureEightMyPi_Step0.py                    #
######################################################################
# Description:-                                                      #
# Seven Segments of Pi Challenge                                     #
# Software for the Game "Figure Eight My Pi"                         #
# Runs on Seven Segments of Pi Games Console                         #
# When PushButton is pressed                                         #
# GPIOs drive Seven Segment Display Segments a,b,c,d,e,f in turn    #
# with each Segment illuminated for 0.5 seconds                      #
# so "PiSeg" appears to go round in a clockwise "Figure Zero"       #
######################################################################
#!/usr/bin/env python    #allows program to be run from command line

import time              #time package allows programmable delays in the software
import RPi.GPIO as GPIO  #RPi.GPIO package allows control of GPIO by software
GPIO.setmode(GPIO.BOARD) #RPi.GPIO package numbers GPIO by their Raspberry Pi Connector pin number
GPIO.setwarnings(False)  #Disables GPIO Warning Messages

GPIO.setup(7, GPIO.IN)   #GPIO 7 is input from Push Button Switch
                         #                                              _______
GPIO.setup(11, GPIO.OUT) #GPIO 11 output illuminates Segment a         |   a   |
GPIO.setup(12, GPIO.OUT) #GPIO 12 output illuminates Segment b        f|       |b
GPIO.setup(13, GPIO.OUT) #GPIO 13 output illuminates Segment c         |_______|
GPIO.setup(15, GPIO.OUT) #GPIO 15 output illuminates Segment d         |   g   |
GPIO.setup(16, GPIO.OUT) #GPIO 16 output illuminates Segment e        e|       |c
GPIO.setup(18, GPIO.OUT) #GPIO 18 output illuminates Segment f         |_______|
GPIO.setup(22, GPIO.OUT) #GPIO 22 output illuminates Segment g             d

poll = .1                           # Define Constant for PushButton 'poll' delay
delay = .5                          # Define Constant for step delay
###################################################
# Start of "Figure Eight My Pi" Program Execution #
###################################################
# start with all Segments off
GPIO.output(11, False)              # a
GPIO.output(12, False)              # b
GPIO.output(13, False)              # c
GPIO.output(15, False)              # d
GPIO.output(16, False)              # e
GPIO.output(18, False)              # f
GPIO.output(22, False)              # g

print "Press PushButton to Start"   # Printed on Raspberry Pi *Python Shell* Window
PushButton = False                  # wait until PushButton has been pressed before starting state machine
while PushButton == False:          # if PushButton has been pressed...
    PushButton = GPIO.input(7)      # Check PushButton input
    time.sleep(poll)                # if not, wait for 0.1 second before checking PushButton again
state = "a"                         # Start State Machine in state "a"

# start of main state machine
while True:                         # while True: means run this loop forever

    if state == "a":                ####### state "a" ######
        print state                 # prints name of state - comment this line out once it is working
        GPIO.output(11, True)       # turn on Segment 'a'
        time.sleep(delay)           # after a delay
        GPIO.output(11, False)      # turn off Segment 'a'
        state = "b"                 # then move to state 'b' which is the next Segment clockwise

    if state == "b":                ####### state "b" ######
        print state                 # prints name of state - comment this line out once it is working
        GPIO.output(11, True)       # turn on Segment 'b'
        time.sleep(delay)           # after a delay
        GPIO.output(11, False)      # turn off Segment 'b'
        state = "c"                 # then move to state 'c' which is the next Segment clockwise

    if state == "c":                ####### state "c" ######
        print state                 # prints name of state - comment this line out once it is working
        GPIO.output(11, True)       # turn on Segment 'c'
        time.sleep(delay)           # after a delay
        GPIO.output(11, False)      # turn off Segment 'c'
        state = "d"                 # then move to state 'd' which is the next Segment clockwise

    if state == "d":                ####### state "d" ######
        print state                 # prints name of state - comment this line out once it is working
        GPIO.output(11, True)       # turn on Segment 'd'
        time.sleep(delay)           # after a delay
        GPIO.output(11, False)      # turn off Segment 'd'
        state = "e"                 # then move to state 'e' which is the next Segment clockwise

    if state == "e":                ####### state "e" ######
        print state                 # prints name of state - comment this line out once it is working
        GPIO.output(11, True)       # turn on Segment 'e'
        time.sleep(delay)           # after a delay
        GPIO.output(11, False)      # turn off Segment 'e'
        state = "f"                 # then move to state 'f' which is the next Segment clockwise

    if state == "f":                ####### state "f" ######
        print state                 # prints name of state - comment this line out once it is working
        GPIO.output(11, True)       # turn on Segment 'f'
        time.sleep(delay)           # after a delay
        GPIO.output(11, False)      # turn off Segment 'f'
        state = "a"                 # then move back to state 'a' which is the next Segment clockwise
```



## Figure Eight My Pi Step0 State Machine

Opposite is “***FigureEightMyPi_Step0.py***”, the starting software for your game. The software starts with all segments ‘off’ then waits until the PushButton is pressed. When pressed it starts a State Machine. The State Machine has 6 states named `a,b,c,d,e` and `f` as shown in the diagram below. In state `"a"` it turns ‘on’ segment ‘a’, waits for a delay of 0.5 seconds and turns off segment ‘a’. It then moves on to state `"b"`. (The line **`print state`** prints the name of the state on the Terminal. When you run the program these lines help you see that the State Machine is flowing in the correct order but should be deleted or “***commented out***” once the State Machine is working correctly as they slow the program down. To “***comment out***” a line of software just type **#** at the start of the line. The whole line will turn red, hence it will be ignored when the program is run). State `"b"` is very similar except it turns ‘on’ and ‘off’ segment ‘b’ and moves on to state `"c"`. States `"c"`,`"d"`,`"e"` and `"f"` are similar but state `"f"` moves back to state `"a"`.

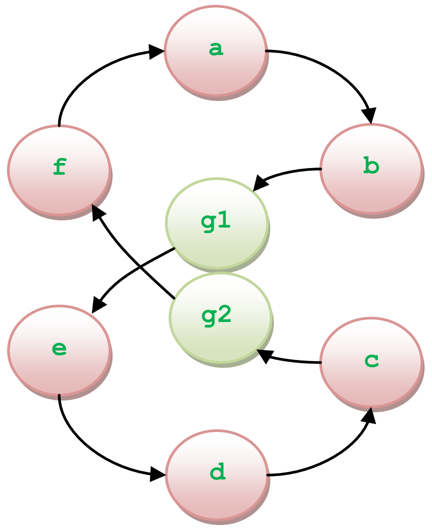


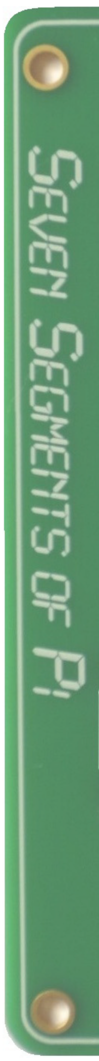
When run, this program illuminates a segment which appears to move around the display in a clockwise direction in a figure zero. This is the “***PiSeg***”!

## Figure Eight My Pi Step1 State Machine

In Step 1 you must change the software to make the ***PiSeg*** go around the display in a ‘figure-of-eight’. This means the State Machine must flow as shown in the diagram below. In this State Machine state `"b"` moves on to state `"g1"` not to state `"c"`. Note that both states `"g1"` and `"g2"` illuminate the middle segment but are different states. State `"g1"` moves on to state `"e"`, whereas `"g2"` moves on to state `"f"`.

When run, the ***PiSeg*** should now move round the display in a figure eight!



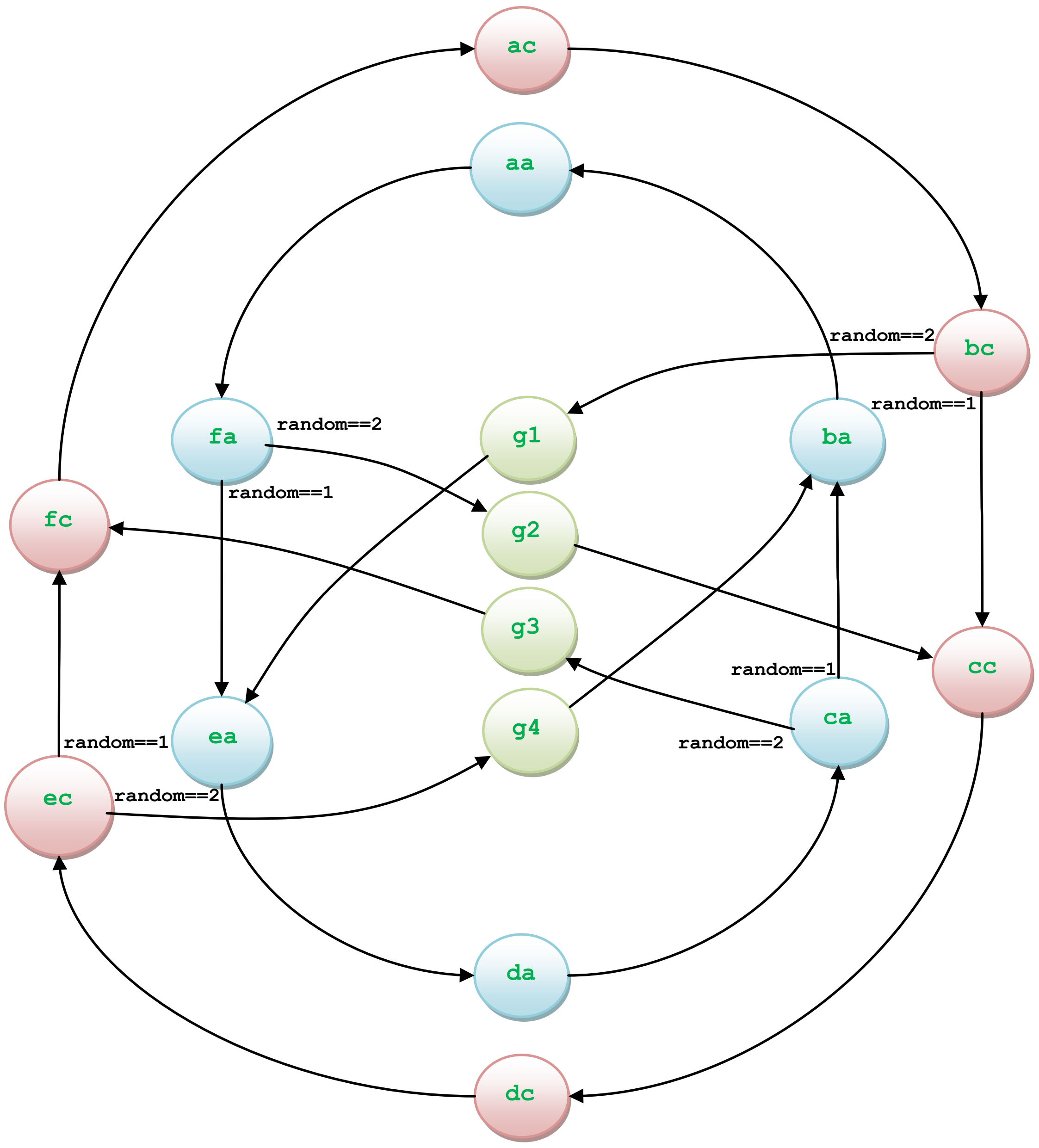


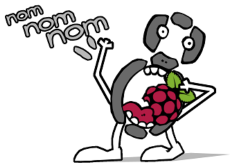
## Figure Eight My Pi Step3 State Machine

The State Machine for the game is completed in Step 3. Below is the State Machine you will need to write. Since the ***PiSeg*** can now randomly either move round the display in a figure zero or a figure eight, sometimes it will be moving round in a clockwise direction and sometimes in an anti-clockwise direction, so for example state `"ac"` is where segment 'a' is illuminated and it is moving in a clockwise direction, whereas state `"aa"` is where segment 'a' is illuminated but it is moving in an anticlockwise direction.

States `"bc"`,`"ca"`,`"ec"` and `"fa"` are the decision states. For instance in state `"bc"` it either moves to state `"cc"` or to state `"g1"`. To have a 50% chance of going across the middle, in state `"bc"` generate a random number between 1 and 2. If this random number is a '2' move to state `"g1"` otherwise move to state `"cc"`.

Note that there are now four different states which illuminate the middle segment, `"g1"`,`"g2"`,`"g3"` and `"g4"`. These need to be different states because the all move on to different states, for instance state `"g1"` moves on to state `"ea"` whereas state `"g2"` moves on to state `"cc"`.





## Figure Eight My Pi Game Extras

If you have completed the "**Seven Steps to Figure Eight My Pi**" you now have a working game! But don't stop there! Have a go at these ***Figure Eight My Pi*** Extras, or come up with your own ideas to make the game even better!

*X1*) Add a musical introduction or sound effect before the game starts and one as the game ends using music or sound effects files on the "***SSPi***" SD Card or downloaded from the internet. Any .wav or .ogg file can be used. You may find the section "**Taking Backups and Transferring Files onto your Raspberry Pi**" useful to if you need to transfer a sound file onto the SD Card.

*X2*) Add 3 levels, Beginner(slow), Intermediate(medium) and Advanced(fast) so that when the program runs the player is prompted with `Beginner?` at which point a long PushButton press starts a slow version of the game or a short PushButton press changes the prompt to `Intermediate?` Similarly at this point a long press starts a medium speed game or a short press moves on to `Advanced?`

*X3*) Create a version of the game with random changes of speed each time the ***PiSeg*** passes through the middle segment.

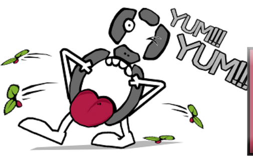
*X4*) Record your own sound effects (e.g. nom, nom, nom?). Download the free Audio Capture software from Audacity. Audacity can be used to first capture your own sound effects from a USB Microphone then to modify it. For instance try speeding up the sound effect or slowing it down. Then export it as a `.wav` file for use in your game.

*X5*) Display image file `PiSeg1.png` on the monitor while the game is playing. Refer to www.pygame.org and related pygame tutorials for how to display images in **Python**. This and other image files can be found on the SD Card but to use them within in your game they must be in same directory as your **Python**.

*X6*) Display image files `PiSeg1.png` and `PiSeg2.png` alternately on the monitor during the game as the ***PiSeg*** moves round the display to give the impression it is eating the Raspberry! And can you do anything with `PiSeg3.png` and `PiSeg4.png`?

*X7*) Create a score board on the display using graphics. For instance you could create a graphical version of a Seven Segment Display as your score board. You will have to create these graphics yourself!

*X8*) Add a "Highest Score" to your scoreboard. Can you find a way to retain the highest score when the **Raspberry Pi** is turned off and on again?



## Troubleshooting Sound Effects

**No Sound?**

The **Raspberry Pi** can generate Analog or Digital Audio. Analog Audio is played via speakers (or headphones) connected to the 3.5mm Analog Audio Output Jack of the **Raspberry Pi**. Digital Audio is sent via the HDMI cable to a TV. The "***SSPi***" SD Card is configured to 'Auto Audio' where audio is automatically set to Analog or Digital depending on the type of monitor connected, so if you are using a VGA or DVI Monitor, audio will normally be set to 'Analog' hence you must connect speakers (or headphones) to hear the sound effects. If you are using an HDMI TV audio will normally be set to 'Digital' so the sound effects will be sent digitally to the TV. If you don't hear any sound it could be that your monitor has confused the **Raspberry Pi** so it is sending audio to the wrong connector! If so, you can ***force*** it to send audio to the right connector using the following commands:-

Open a Terminal Window using
**Start>Accessories>LXTerminal**
Then to force Analog Audio type command

```
sudo amixer cset numid=3 1
```

Or to force Digital Audio type command

```
sudo amixer cset numid=3 2
```

Or to revert to Auto Audio type command

```
sudo amixer cset numid=3 0
```

If when you switch off your **Raspberry Pi** you select
**Start>Logout>Shutdown**
and wait until the **Raspberry Pi** has "Halted" before removing power, the above audio selection will be saved so you won't need to type the command in again.

**Clicks but no Sound Effects?**

If you hear a click but no Sound Effect **Python** has not found your sound effect file. Check that the sound effect file is in the same directory as the **Python** file and that it has exactly the same file name as called by your program.

**Distorted Sound?**

If when you play a sound effect the sound is distorted you may have to increase the audio buffer size in your Python program. The line in the program

```
pygame.mixer.pre_init(44100,-16,2,512)
```

sets the buffer size to 512 Bytes. This value should be OK for your sound effects but this value could be changed to values as small as 128 or less, or as big as 2048 or more. If this buffer size is too small the sound effects may be distorted, so try increasing the buffer size to, say 640.

```
pygame.mixer.pre_init(44100,-16,2,640)
```

**Delayed Sound?**

If when your program plays a sound effect there appears to be a delay before you hear the sound you may have to *decrease* the audio buffer size in your program to say 448.

```
pygame.mixer.pre_init(44100,-16,2,448)
```

…but of course it might start to distort!
Experiment to find the smallest value that doesn't distort.

## SSPi SD Card Directories

The main directories on the "***SSPi***" SD Card used in the Seven Segments of Pi Challenge are:-

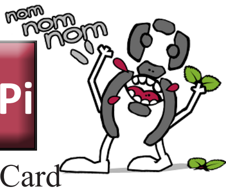
```
/home/pi/share/PiDice
/home/pi/share/PiStopWatch
/home/pi/share/FigureEightMyPi
```

But there are also backup copies of all original files in:-

```
/home/pi/backup/PiDice
/home/pi/backup/PiStopWatch
/home/pi/backup/FigureEightMyPi
```

To restore the original files and thus re-start the Challenge, copy all 3 backup directories into the share directory, overwriting the existing directories.

***WARNING - THIS WILL DESTROY ANY SOFTWARE YOU HAVE WRITTEN!***



All program files required for the ***Seven Segments of Pi*** Challenge are included on the "***SSPi***" SD Card however it is important that you take regular back-ups of your programs files in case the SD Card gets damaged or lost. You may also want to transfer files from a PC onto your SD Card, for instance to add new sound effect files. The simplest way to do this is to use a **USB Memory Stick**. A more advanced way to do so is to do set up a **Remote Login** from your PC to your **Raspberry Pi**. Both methods are described below.

**USB Memory Stick**

These instructions assume you have a USB Keyboard and Mouse attached to your **Raspberry Pi** and you are transferring files to or from a Windows PC.

On your PC - Insert a USB Memory Stick and create a new directory on it called **RPiBackup**.
On your **Raspberry Pi** – Start>File>Accessories>File Manager
    Browse to the directory with files to transfer e.g. /home/pi/share/PiDice
    Unplug the USB Keyboard from your **Raspberry Pi** (but leave USB Mouse connected)
    Plug the USB Memory Stick into your **Raspberry Pi**
    A pop-up message will say "Removable medium is inserted"
    Use Mouse to select "**Open in File Manger**" <**OK**> This opens a 2nd File Manger Window
    Browse to directory **RPiBackup**
    In 1st File Manager Window select the files you want to back-up
    e.g from directory /home/pi/share/PiDice select your completed program **PiDice_step7.py**
    Select **Edit**>**Copy**
    Select 2nd File Manger Window and **Edit**>**Paste**
    Close 2nd File Manger Window
    Remove the USB Memory Stick
    Reconnect the USB Keyboard to the **Raspberry Pi**
On your PC - Insert the USB Memory Stick and browse to directory **RPiBackup** to see your files.

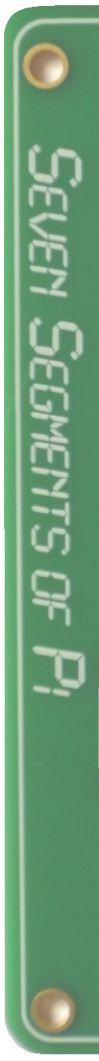
**Remote Login**

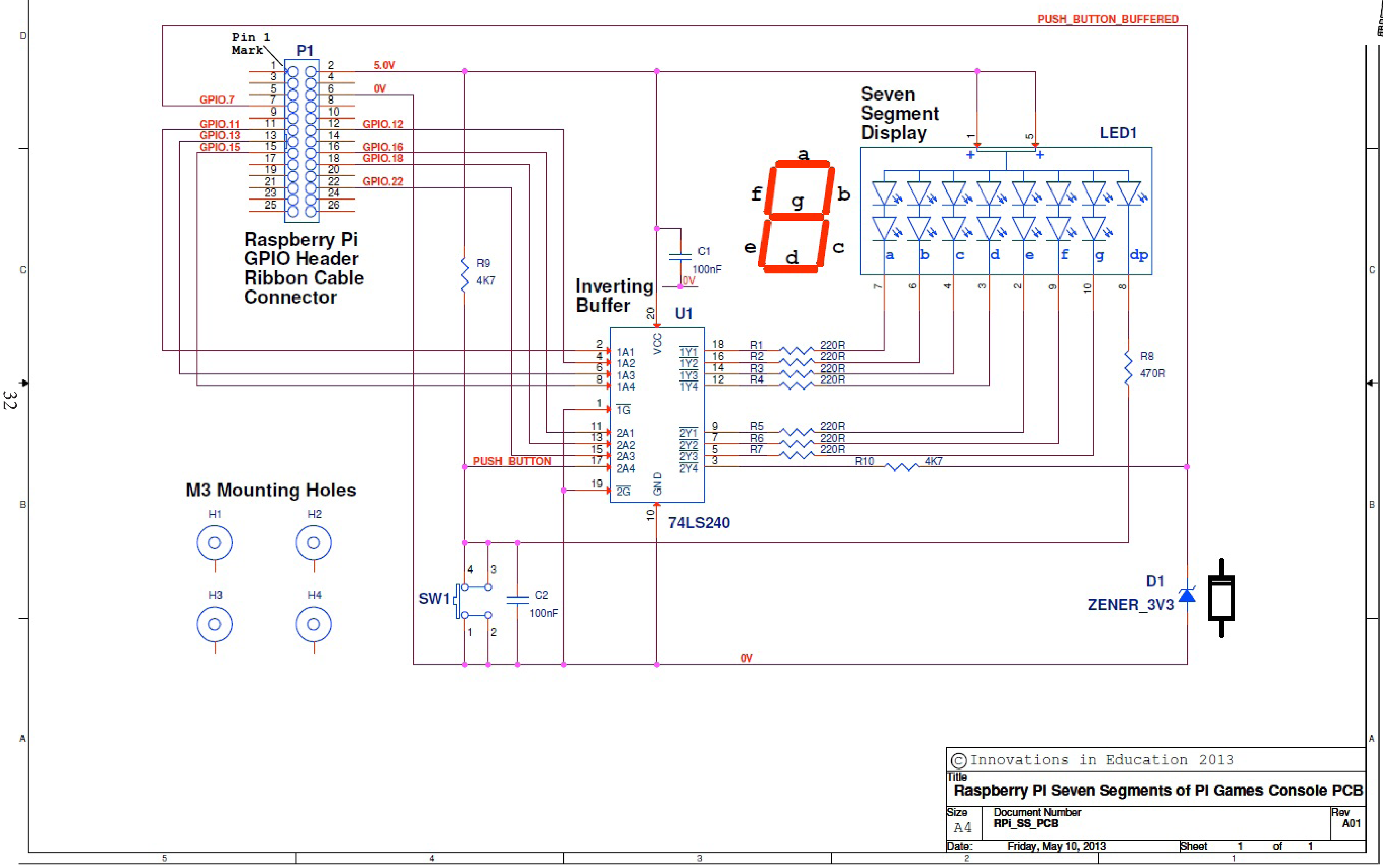
These instructions assume you have a wired Ethernet connection to your **Raspberry Pi** and you are transferring files to or from a Windows PC connected to the same Ethernet Router as your **Raspberry Pi** (either by Ethernet Cable or WiFi). The instructions also assume you are using the "***SSPi***" SD Card which has Samba enabled to allow Remote Login and is configured to give your **Raspberry Pi** the name "***RPi***".

On your **Raspberry Pi** – Attach an Ethernet Cable from your **Raspberry Pi** to your Ethernet Router
On your PC - Download the program **putty.exe** from www.putty.org
    Double click on **putty.exe** and select <**Run**> to run the program
    Ensure Connection Type is set to **SSH**
    In "Host Name (or IP address)" box enter **RPi <open>**
    An RPi-Putty window should open
    (First time you run it you will get a PuTTY Security Alert, select **<Yes>**)
    and, if it finds your **Raspberry Pi** via your Ethernet Network,
    it will display the normal **Raspberry Pi** login prompt, so login as usual

```
login as: pi
pi@RPi's password: raspberry
pi@raspberrypi ~$
```

    *You are now have a Remote Login connection to your* **Raspberry Pi**
On your PC – Start>My Computer>My Network Places
    Select **share on raspberrpi server (Samba Raspberry Pi) (RPi)**
    You should now be able to see the directory **\\RPi\share.**
    (This is the directory **/home/pi/share** on your SD Card)
    Browse to the directory with the files to transfer e.g. **\\RPi\share\PiDice**
    Select files to back-up e.g. **PiDice_step7.py** and select **Edit**>**Copy**
Browse to your back-up directory on your PC and select **Edit**>**Paste**





©Innovations in Education 2013

Title: Raspberry PI Seven Segments of PI Games Console PCB

| Size | Document Number | Rev |
|---|---|---|
| A4 | RPi_SS_PCB | A01 |

Date: Friday, May 10, 2013 Sheet 1 of 1

The ***Seven Segments of Pi*** connects to the Raspberry Pi via a Ribbon Cable plugged onto the Raspberry Pi's `P1` Connector.

The electronics consists of three main components:-

- ***A Red PushButton Switch as the Control***
- ***A Seven Segment LED as the Display***
- ***A Display Driver Integrated Circuit (IC)***

The Raspberry Pi controls the ***Seven Segments of Pi*** via 8 GPIO (General Purpose Input/Output) signals on its `P1` Connector. GPIO are signals that can be configures as Inputs or Outputs under software control. The ***Seven Segments of Pi*** uses one GPIO as an input so the software can check if the Red PushButton Switch has been pressed and seven GPIO as outputs so the software can control which segments of the seven segment display are illuminated.

**The *Seven Segments of Pi* Circuit Description**

You don't need to understand the electronic circuit to assemble your kit or write your software however the better you understand the electronics the easier you will find it to fix any assembly errors. It may also help if you are thinking of designing your own electronics to connect to the Raspberry Pi!

Opposite is the ***Seven Segments of Pi*** circuit diagram.

**`LED1` - Seven Segment LED Display**. A seven segment display is a device that integrates 7 Light Emitting Diodes (also known as segments) into a single device to create the familiar Seven Segment Display. These displays allow single digit numbers from 0 – 9 to be displayed by illuminating the correct combination of the 7 LEDs. The 7 segments are traditionally referred to by lower case letters a,b,c,d,e,f,g as shown on the Circuit Diagram. So for instance the number '1' would be displayed by illuminating segments b and c only.

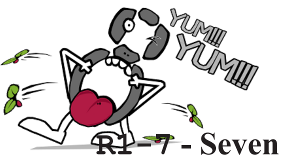
I said that there are 7 LEDs. In fact if you look at the Circuit Diagram, it appears to have 15 LED's! This is correct since for this display each segment is actually 2 LEDs in series which makes each segment brighter. There is also a 15th LED labelled "dp". This is the Decimal Point LED which is in the bottom right corner of the display. On the ***Seven Segments of Pi*** this dp LED is wired to illuminate whenever the Red Push Button Switch is pressed.

The display is known as a "Common Anode" version because the anodes of all LEDs are connected together within the device. 5 Volt power is connected to the common anode pins 1 & 5. The cathodes of each LED are however available on separate pins thus allowing us to drive each segment individually. So if for instance we drive pin 7 to Logic '0' segment 'a' will illuminate.

The ***Seven Segments of Pi*** uses the Kingbright SA15-11SRWA 38mm display - the largest display that can be driven from a 5V power supply. The pins on the display are numbered from 1 (top left when looking at display from above) to 5 (top right) then 6 (bottom right) to 10 (bottom left) but note that on the circuit diagram the symbol shows the pins in a logical order not in the order they appear on the physical part. The Data Sheet can be downloaded from www.kingbright.com

**`U1` - Display Driver IC. `U1`** 'drives' the Seven Segment LED Display. It is a 74LS240 Octal Inverting Buffer in a 20 pin Dual In-Line (DIL) package. 7 of the 8 Buffers drive the Display by providing 3 main functions. Firstly they convert the 3.3V Logic signals from the Raspberry Pi's GPIO to 5V Logic signals needed to drive this Seven Segment LED Display. Secondly they provide the 7mA drive current needed to drive each segment of the display. Thirdly they invert the GPIO Logic levels so a GPIO set to Logic '1' (i.e. 'True' in your **Python** software) will output a Logic '0' to the LED display which turns the LEDs 'on'. Hence in your **Python** 'True'='on' & 'False'='off' which is easier to remember! The 8th Buffer is used for `SW1` as described below. The Data Sheet can be downloaded from www.ti.com

**`SW1` - PushButton Switch**. `SW1` has 4 pins. Pins 1 & 2 are connected within the switch and pins 3 & 4 are also connected within the switch. When the switch is pressed pins 1 & 2 are connected to pins 3 & 4 thus making the input to the Display Driver Buffer 0 Volts (i.e. Logic '0'). When the switch is released `R9` pulls the Buffer input to 5 Volts (i.e. Logic '1'). The Buffer inverts this signal so when the PushButton is pressed, the GPIO sees a Logic '1'. Hence once again in your **Python** Software 'True'='on' & 'False'='off'.



**R1-7 - Seven Segment LED Current Limiting Resistors.** R1-7 set the brightness of the Seven Segment LEDs. R1 sets the brightness of Segment 'a', R2 sets Segment 'b', etc. The Kingbright display could be set to anything from about 2mA (dim) to 30mA (very bright). R1-7 are 220 ohms which sets the LED current to about 7mA.

To calculate the approximate current for different values of current limit resistor use the formula:-

$I_{LED} = (V_{SUPPLY} – V_{LED}) / R_{LIMIT}$

Where

LED Anode Voltage $V_{SUPPLY} = 5V$

Voltage drop across the LED $V_{LED} = 2 \times 1.75V$ (approx)

Current Limit Resistor $R_{LIMIT} = 220$ ohms

Current through LED $I_{LED} = 7mA$ (approx)

**R8 - Decimal Point LED current limiting resistor.** R8 sets the brightness of the Decimal Point LED. Note that the Decimal Point is only a single LED hence $V_{LED} = 1 \times 1.75V$ (approx) so $R_{LIMIT} = 470$ ohms make $I_{LED} = 7mA$ (approx)

**Z1 & R10 - GPIO Input Protection.** The Display Driver IC when driving a Logic '1' will output a voltage up to 5V however Raspberry Pi GPIO should not be driven with voltages greater than 3.3V! The Zener Diode D1 prevents the GPIO voltage from exceeding 3.3V. (See below for a further explanation of Zener Diodes). Also, there is a chance when you are developing your software, you might accidentally make the PushButton Switch GPIO an output. If so, resistor R10 will prevent any damage to the GPIO.

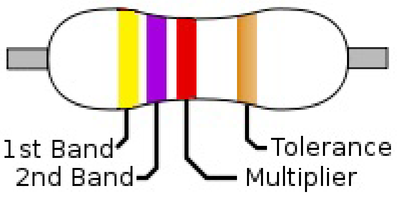
**Zener Diodes.** Normal Diodes conduct in the 'forward' direction (when the Anode Voltage is greater than the Cathode Voltage) but don't conduct in the 'reverse' direction (when the Cathode Voltage is greater than the Anode Voltage). Zener diodes conduct in the same way in the 'forward' direction, but they also conduct in the 'reverse' direction, but only if the voltage at the Cathode is greater that the voltage at the Anode by a value exceeding the 'breakdown voltage'. The Zener Diode used here has a 'breakdown voltage' of 3.3V. Hence, if the voltage on this GPIO reaches 3.3V, the Zener Diode will start to conduct, thus preventing the voltage from exceeding 3.3V.

**C1 - Decoupling Capacitor.** This capacitor filters any 'noise' on the 5V power rail so the Display Driver IC has a 'clean' power rail.

**C2 - Debounce Capacitor.** When the PushButton Switch is pressed the voltage on the input to the Display Driver IC will fall from 5V down to 0V. C2 ensures that if the switch 'bounces' after the voltage has dropped below the Logic '0' threshold of the Display Driver IC it will not immediately rise above the Logic '1' threshold. This is known as a 'debounce' circuit.

**5V Power Rail.** The electronics is powered by the 5V power rail from the Raspberry Pi. With all segments illuminated, including the decimal point, the ***Seven Segments of Pi*** consumes about 80mA hence your Power Supply must be able to supply this extra current. It is therefore recommended that you use a Power Supply that can supply 5V at a current of 1A (1000mA) or more.

**Resistor Colour Coding**



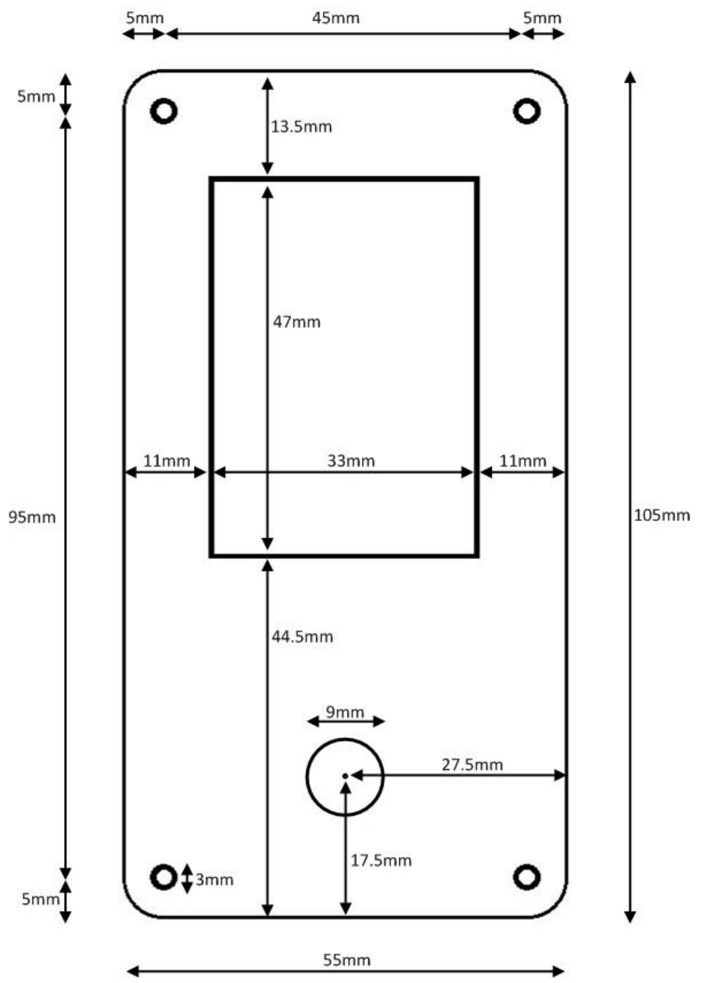
The resistors used in the kit have 4 coloured bands indicating their value in Ohms where Black=0, Brown=1, Red=2, Orange=3, Yellow=4, Green=5, Blue=6, Purple=7, Grey=8, White=9. The first 2 bands give the first 2 digits of the value with the 3rd 'Multiplier' band giving the number of zeros, hence Yellow, Purple, Red = 4700 Ohms or 4.7kOhms, often written as 4k7. The 4th band indicates the tolerance. Gold=5% hence the actual value of any resistor marked 4k7 might be anything from 4.7k -5% (4.465kOhms) to 4.7k +5% (4.935kOhms)

## Front Panel - DIY or Perspex

Like the Raspberry Pi, the ***Seven Segments of Pi*** is supplied without a case but you can either make your own Front Panel or buy the ***Seven Segments of Pi* Perspex Front Panel Kit**.

### DIY Front Panel

To make your own Front Panel use stout cardboard or similar material cut to the dimensions shown below and decorate it with your own design!



To fix your DIY Front Panel to the ***Seven Segments of Pi*** you will need the following:-

- ***4 x M3x16mm Panel Head Screws***
- ***4 x M3 Nuts***
- ***4 x M3 Washers***
- ***4 x M3x10mm Spacers***

### Instruction Manual Updates

Whilst every effort has been made to ensure these instructions are comprehensive and accurate it is possible there may be errors or omissions. Please let us know of any such errors or omissions via the web site www.SevenSegmentsOfPi.com.
Your help in doing so is very much appreciated!

We will publish on the web site any corrections or updates as we become aware of them.

### Perspex Front Panel

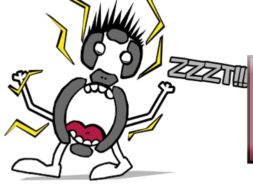
The ***Seven Segments of Pi* Perspex Front Panel Kit** is supplied complete with all screws, nuts, washers and spacers to attach it to you ***Seven Segments of Pi***.

**T.B.A**

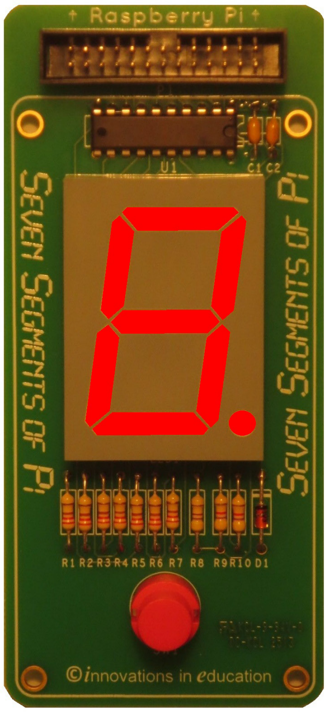
### Acknowledgements

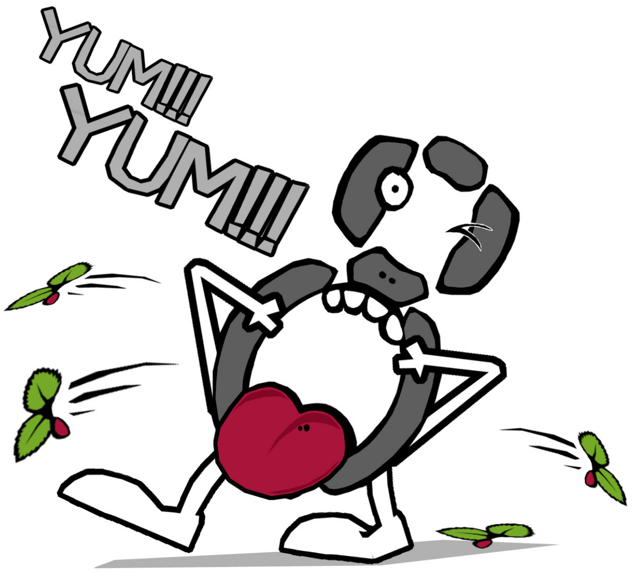
I would like to express my gratitude to everyone who has contributed to the ***Seven Segments of Pi***.
In particular I would like to thank:-

**Advanced Layout Solutions**, Berkshire UK, for the Printed Circuit Board Layout.
**Merlin Circuits**, Deeside UK, for the Prototype Printed Circuit Boards.
**Pupils and Teachers of Kennet School,** Thatcham, UK, for helping develop the software challenges and 'test driving' the prototype kits.
**Matt Hunt** and **Paul Blenkhorn** for the graphics.
**Alastair Bryers** for help with Linux, ***Raspbian*** and NOOBS.



# Seven Segments of Pi Kit Contents





## Kit Contents

- *Seven Segments of Pi PCB*
- *38mm Seven Segment Display*
- *Display Driver Integrated Circuit*
- *PushButton Switch*
- *Resistors (7 x 220R, 1 x 470R, 2 x 4k7)*
- *Capacitors (2 x 100nF)*
- *Zener Diode*
- *Connector*
- *Ribbon Cable*
- *Stick-on Feet*

## Kit Version 'A'

- *Software included on "SSPi" SD Card*

## Kit Version 'B'

- *Software downloaded from Web Site*

## Tools Required

- *Soldering Iron*
- *Solder*
- *Small Pliers*
- *Wire Cutters*

## Required but not included

- Raspberry Pi® *Computer*

## Also Required

- *Your Own Imagination!*

***...and this could be your first step to becoming one of the next generation of Computer Games Designers!***

innovations in education

**www.SevenSegmentsOfPi.com**