Final Project Documentation:

The Bare Bones Cheapskate Raspberry Pi

The Basic Rundown:

The goal of our final project was to take the $35 Raspberry Pi computer—which essentially just amounts to a useless paperweight when not accompanied by all the external hardware and equipment that is normally needed to make it a viable tool (monitor, cables, keyboard, mouse, etc.)—and turn it into a fully functioning computer in its own right (which it probably should've been right from the start). The basic premise here is that the Raspberry Pi should be willing and able to accommodate all users—even if they don't necessarily have all the equipment/money that might be needed to take *full* advantage of the raspberry pi's capabilities and uses. The Raspberry Pi should continue limping along as best it can, regardless of how many pieces of external hardware go down—so long as it is physically possible for the Pi to carry out the user's instructions, it should (and will now) continue to attempt to do so to the best of its abilities.

The simplest and cheapest possible means of input and output would just be two wires (or any conductive material) to use as a button to input Morse Code into the assigned button pin on our Raspberry Pi and an LED to allow the Raspberry Pi to send info back to the user using Morse Code as well. This was the most basic structure of our code at its most rudimentary level—however, we wanted to make sure that we allowed the user to make use of whatever tools he had available to him, and to give him the ability to adjust input and output settings to best suit his/her needs.

We wanted to create a Pi that would be able to function under *many* different circumstances—and would continuously adjust its behavior based on the actions of the user (as well as what the Pi could potentially *detect* on its own). In other words our Pi has the ability to make small adjustments on its own and won't just crash with long error messages the second a single wire comes loose—instead it will simply notify the user of the potential change and then without delay will continue to attempt to perform the task that has been set before it to the best of its ability under the new circumstances/limitations.

In addition our expectation is that it is entirely possible that the user has no means of executing this python code we've created on their own (since the only people who are likely to want to use code that allows the user to input and output without a mouse, keyboard, and/or monitor are likely to be those who simply don't have access to these things in the circumstances that they're in), therefore we decided to create a shell script and various structures within the Raspberry Pi itself (outside of our python code) that would simply execute our python code at the start of every session (in case the user has no way of doing this on their own). I will detail how this was done as well as it is relatively simple.

Getting our Python Code to Execute Automatically each time the Raspberry Pi is turned on:

This was a relatively simple process—first we created an empty text file and then we copied the following into it (since we would need to repeat this whole procedure for every single pi we were working with):

```
POINTLESS="stuff"

echo $POINTLESS

HOPEFULTEST="harmless2.py"

echo $HOPEFULTEST

#sudo python $HOPEFULTEST

sudo python3 /home/pi/pyAtStart/$HOPEFULTEST
```

This shell script first does some pointless stuff just for show and then runs our python code harmless2.py saved within the variable "$HOPEFULTEST."

Then we changed our permissions by entering chmod 777 bmanScript4 to allow us to use it.

Finally we then simply went to our ~ using cd ~ if not already there and then began altering .bashrc by entering sudo nano .bashrc into the terminal. Then we simply scrolled down to the ***very*** bottom and added the following line to the very end ./bmanScript4

…this of course would set us up so that our python code would automatically run at the start every time we reset our pi. In order to shut this off if we no longer want it happening anymore, we can simply comment out that last line we added making it #./bmanScript4

…until we decide we need it again at which point we can, of course, simply uncomment it.

Adding MP3's (Music) to the Approriate Location:

This is obviously very straightforward: just create the following directories/subdirectories as needed:

/home/pi/Desktop/finalProject/music/Electronic/Chillout

…just create any empty folders with those names in that order, and then put all your MP3s into that final subdirectory called "Chillout":

The Parts List:

Bare Minimum:

- An Led (ours are just one's we've harvested from Christmas tree lights)
- A couple of paper clips (…or any other random bit of metal that you can use to touch our button pin to some ground with)

Technically this is all that is needed to make our code technically viable—with just these two things alone or Pi will be able to output messages to the user via Morse Code and enter input into the pi via Morse Code by touching the button pin to ground using any random bit of metal… however, to make this sort of device truly viable and useful you'll probably want at least the following:

More Viable/Useful Parts List (in addition to bare minimum):

- Any random assortment of wires you might have lying around to hook things together
- A standard character LCD screen hooked up to an I2C backpack
- Headphones or speaker to listen to MP3's you play via Raspberry Pi

Other Useful/Helpful things to Have:

- A pack of sugar-free gum with at least one stick left
- Some breathable non-stick athletic tape
- A few rubber bands
- A breadboard with a button in it can be useful for entering clean, accurate, Morse Code.
- Jumper cables are helpful but not necessary
- It wouldn't hurt to have a couple alligator clips to extend the length of various wires and a few extra paper clips to make quick, easy connections if need be

The User's Guide:

When the user first turns on our Pi the python code we've created will start up on its own automatically. This may take a minute or two—just be patient.

As has been detailed in previous labs, our python code is designed to function with an IC2 backpack soldered onto any standard character LCD display—by default we use a 2 line LCD screen which can display 16 characters on each line but we give the ability to adjust this to whatever suits their need (within reason). Messages will begin displaying on your LCD screen (as well as being output via Morse Code LED blinks if the user has turned that setting on—however this method is rather tedious and not a very realistic method of getting accurate info within a reasonable time frame). In order to rapidly skip over these messages displayed by the Raspberry Pi simply hold down your button (or touch your button pin to ground for a second or two with some conductive material). To switch to keyboard input mode, simply skip through these messages until the LCD screen prompts you to begin entering Morse Code. At that point the user can simply enter any button presses that are not considered a valid selection (just one or two dots or dashes is sufficient) by touching your button pin to ground a couple times with any conductive material-- or simply wait until the Morse Code input detection times out (usually about 60 seconds) if you don't know what to enter.

When either of these things happens, our code interprets this as a potential change in circumstances—either the user doesn't have the ability to enter Morse Code input via button presses, *or* they don't have the ability to enter Morse Code via button presses *accurately*. This leads our Pi to conclude that the user would be better suited with using a keyboard instead (if they have one). It therefore will then display a message to the user allowing them to either switch to keyboard input mode (by entering "k" on their keyboard) or entering other letter key to allow them to continue to attempt to use Morse Code inputs for now. It should be noted that even though Morse Code is quite difficult to enter accurately, this circumstance I just described will *almost never* happen by accident. This is because our code is specifically designed to be *extremely* accommodating and flexible in accepting user input—usually so long as *any* keystrokes/input the user entered might be considered a valid input, our python code will attempt to meet the user half-way by interpreting the users input as best it can and attempting to carry out the user's instructions to the best of its abilities (even if 90% of the user's request is pure gibberish).

If instead of skipping over these messages, the user decides to allow them to display, our user will be presented will a list of potential options to choose from, in the form of a series of menus and sub menus which is structured in the following manner:

Our code will do the following things successfully (all of which we have tested thoroughly and seem to work well both *with* and *without* a keyboard/monitor/mouse based on our testing—those items that have *not* been fully tested or were never fully completed will be marked as such but everything listed here has been tested extensively—except for those things highlighted in **red**):

1) Create a file and enter text into it
2) Allow the user to select a file and display its contents to allow the user to "read" it
3) Allow the user to perform an online search for information on a specific term, first using Wikipedia and then using the search engine duckduckgo
    a. This will basically just display a brief summary of a few quick, concise sentences on the subject we searched for. However, if no Wikipedia article in this subject exists we will only get output from our duckduckgo search engine.
4) Allow the user to check their emails by displaying them to the user
5) This allows the user to send email messages
6) This allows the user to execute terminal commands—for safety's sake, we decided to limit the user's ability to execute terminal commands to within a specific list. Our code is constantly struggling to find the perfect balance between safety, convenience and flexibility/freedom (in terms of giving the user whatever they want/ask for even if the user's request might not be a good idea)—but I think something like this is probably the best balance we can come up with.
    a. List all files in our current directory and display them to the user
    b. Display the current time and date to the user
    c. Allow the user to create a new directory
    d. Allow the user to change their default directory by allowing them to select a subdirectory within their main directory—this is the only thing here I haven't really *fully* tested enough to determine what problems it might potentially cause—all I know is that it works sufficiently well in the limited tests that I've run so far—therefore, I've decided based on the testing that I've done that the benefits from this outweigh the costs. The user needs to be able to change the default folder he's in, in order to keep things relatively sorted/organized/contained—this will allow them to do so, and the benefits outweigh the dangers/problems it could potentially cause.
7) This allows the user to play music. It first displays a list of all songs in the directory where we decided to store them—in our case we decided on storing all of our MP3's in the following location: /home/pi/Desktop/finalProject/music/Electronic/Chillout. The strange naming scheme here is due to the fact that in earlier iterations of our code we had wanted to allow the user to select between various genres and subgenres—however, we didn't have enough time to implement this idea, therefore you will occasionally see these strange, superfluous remnants of this attempt scattered throughout our code (mostly commented out now)—I just don't have the heart to get rid of them completely. Regardless, if the user wants to plug a different set of MP3's in, this is the location of the directory where they should do so. Any MP3 you put into that folder will be incorporated smoothly into this code and will result in a different list of songs being displayed to the user (obviously). In addition to allowing the user to play music from an MP3 library, it also allows the user to gradually, incrementally, bit by bit adjust the audio settings in the following ways (while the MP3 is currently playing):
    a. U= volume Up
    b. D= Decrease volume
    c. S= Slow down audio
    d. I= Increase audio speed
    e. P= Toggle Pause/Play (this option can be a little bit wonky sometimes but no huge problems are generated by it therefore we left it in)
    f. Q= Quit Playing Music
    g. After hitting Q, we have added a *couple* relatively new features that give the user the option of playing the next song on the playlist by hitting "N" or going *Back* to the previous song using "B". This is now a fully functional easy to use method by which the user can continually play music through many songs for long periods of time.

8) This allows the user to quit the program—this is *essential*: it's very important to give the user the ability to exit our code if need be, especially since (due to their probable lack of input/output devices) they very likely have no means of exiting our code in any sort of conventional manner outside of selecting this menu option
9) This opens up a sub-menu which allows the user to change various important settings mainly related to adjusting the manner in which inputs are accepted and outputs are displayed. It should be noted for the sake of sanity that I've placed all the really important input and output settings into a single global array that looks like this: lcdGlo = [(2),(16),(0.14),(0.35),(21),(0),(""),(16)]—where the respective members of the array are represent the following variables: number of lines per LCD screen=0, characters Per Line of LCD screen=1, time Unit length=2, text Display Speed=3, led Pin = 4, blink LED to Output Morse Code (on/off state)=5, subdirectory=6, button pin=7 (in terms of their location within the array).
    a. **Toggle Led Morse Code output mode:** selecting this option will turn the LED blinking output from the Raspberry Pi *on* if it is currently off, and *off* if it is currently on. Turning this on will allow the Pi to display outputs to the user via Morse Code using LED flashes—however this will make every output the Pi tries to display to the user take *much* longer to do so. Shutting this off is probably a good idea for convenience's sake. It should be noted that although I have extensively tested this and have every reason to believe that the outputs being displayed by the pi are correct, no matter how many times I watch this I simply cannot know with absolute certainty that this is actually working correctly—it's very easy to get lost in the blinks and lose track of what letter/word you're in while at the same time trying to retain how many dots and dashes are in the current letter you're viewing while at the same time trying to distinguish between the slight differences in timing of every single blink and pause—in the end I honestly don't think Morse Code output via blinking is a very realistic way of conveying info to the user. Morse Code input *from* the user however, I've found to be surprisingly useful and easy to use.
    b. **Change Led GPIO pin:** this just allows the user to change the default pin from the current one to some other number GPIO pin. Basically, I just want our code to be very flexible in terms of allowing the user to select from an *extremely wide range* of input and output display options. The most likely reason why someone might do something like this is if the part of their pi where the default LED pin is normally located is broken or obstructed-- or is currently in use for other things. It should be noted that we can also change our button pin number—but this option is only offered if we incorrectly enter something into the main menu (or fail to enter anything at all)—if this happens our python code draws the conclusion that either the user has their button plugged into a different GPIO pin than the current default (or they don't have a button at all) and therefore in addition to offering them the choice of switching to keyboard input mode they also offer the alternate option of changing their default GPIO pin number for their button pin to something else. From the testing I've done, this seems to work fairly well.
    c. **Change LCD screen characters per line:** this allows us to change our default 16 character display to something larger to fully make use of a larger LCD screen (since we have multiple sizes). Unfortunately although this works quite well we have not quite finished the menu option that allow us to *increase the number of lines* that our lcd screen would display to—therefore we decided to scrap that menu option for the time being (although some remnants of this still exist scattered throughout our code).
    d. **Change time unit length:** This I'm really happy with—this allows the user to adjust the standard time unit length to something slightly shorter or longer based on the preferences of the user. Changing this allows the user to send and receive Morse Code blinks --and to distinguish between things like dashes and dots—based on their own personal preferred conversational pace. If they feel like the differences between dashes and dots are too difficult to distinguish, they can simply increase the default length of a time unit to something slightly longer.
    e. **Change text display speed:** This changes the speed at which text is displayed via the LCD screen—making this number larger causes text to display more slowly—making it smaller causes text to display more rapidly. Any good system should be flexible, and should continually strive to accommodate the preferences of the user—therefore for a piece of code which is all about giving the user the greatest possible freedom in displaying/receiving input and output, something like this is obviously essential.

The Wiring:

The wiring we used for our final project is designed to be extremely simple and extremely flexible. In fact we essentially included a diagram of our wiring within the code itself (although the ASCII art we created for this diagram was warped when we transferred it over from our python code into this documentation/report it looks quite nice within the code itself). Again, this was done for the simple reason that we can't know for certain that the user has access to instructions that would show where things should be plugged in (or would even know where to go to *look* for them). If the typical Raspberry Pi user is most likely going to be a relative beginner at working with computers, we shouldn't assume they know too much, and we shouldn't assume they have necessarily have full access to all the information they might need—after all, if their position is so desperate, and their current options are so limited/constrained that they would consider using Morse Code input/output as a viable solution to their predicament then it seems to me fairly reasonable to conclude that we should not overestimate their current access to information/solutions—it may very well be that the only info they necessarily have is within the code itself. Therefore, even if they don't have a keyboard, so long as they have a monitor and a mouse (or a little monitor with a touch screen) they should be able to scroll through and see this diagram within the code itself.

Again, our wiring is simple-- just follow the labels on the LCD screen I2C board in the following way: simply connect ground to ground, SDA to SDA1, SCL to SCL1, and vcc to 5 volts. Then for your button pin, the default is set as GPIO16 and the Led pin is set to GPIO21 by default but you can change the button pin and led pin locations from within the code if you want.

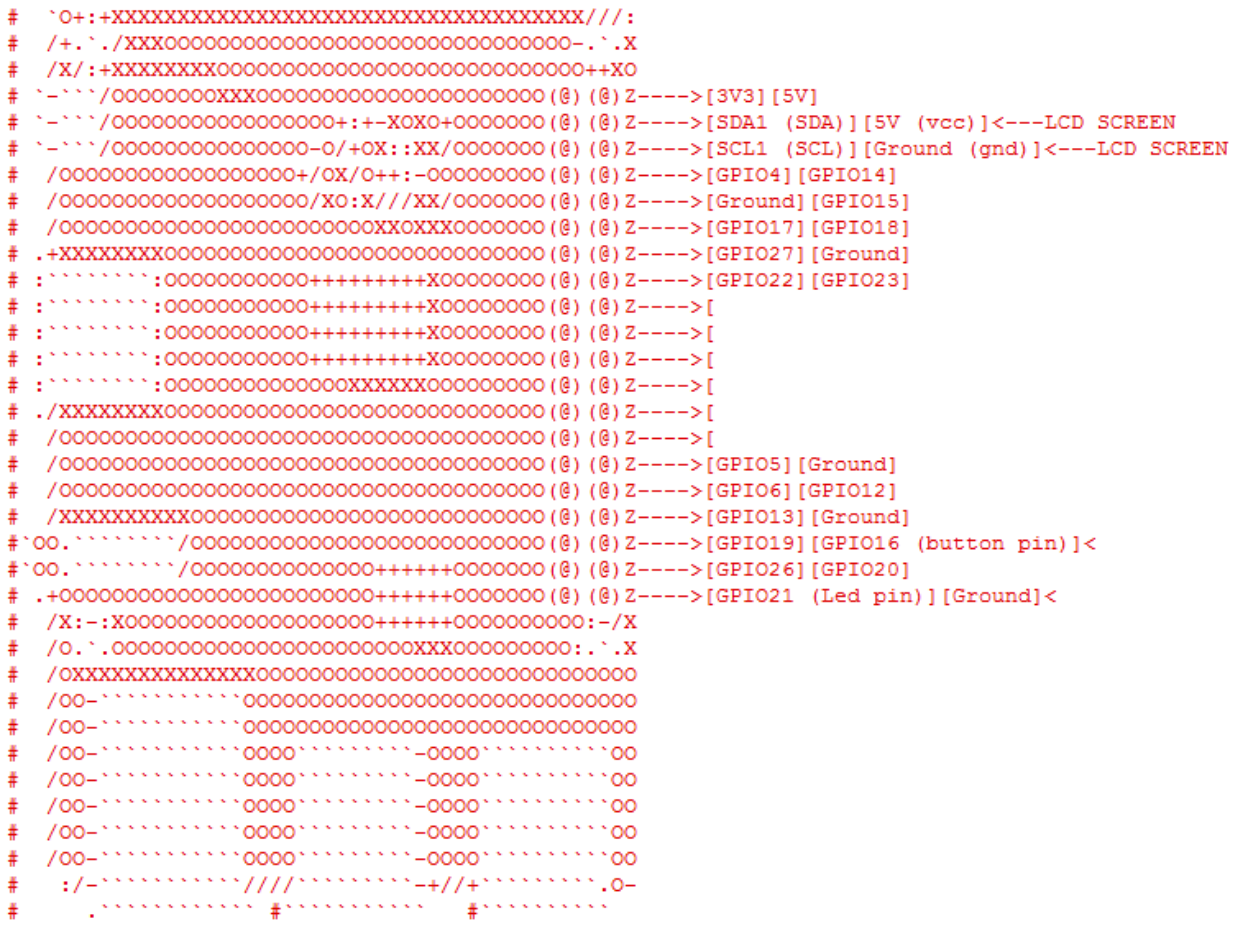


Figure 1. This shows the wiring we used for our Pi, which we displayed by creating an ASCII art diagram of our Raspberry Pi within the comments of our code itself

The whole point of this project is to sort of simulate the sort of "Apollo 13"-esque situation where your resources are extremely limited and then try to figure a way out of it. Obviously we have all the equipment we need if we really wanted to make our project look really pretty-- but that kind of defeats the purpose of the project as a whole—as Ken Mattingly told the NASA technicians as he was getting ready to run the simulator that would give him an idea of what the Apollo 13 crew was going through so that he could attempt to come up with a viable solution given their current situation: "don't give me anything that they don't have up there."

When this project was initially proposed we spent a fair amount of time in our proposal discussing previous attempts that had been made to produce some sort of "bare minimum, adequately functional computer" such that

every single person, even if they be the resident of some impoverished third world nation, would have access to their own computer-- with our discussion focused specifically at times on the OLPC project and their ambitious $100 laptop design. The two biggest constraints to any sort of positive change happening in any part of the third seem to be (obviously) money and infrastructure. The main problem with the $100 laptop is that it seems to be attempting to remedy these constraints by refining their initial mistake instead of recognizing that their current goals might clash somewhat with the constraints of their situation, and altering goals/strategy so that these limitations are taken into account. Although the designers of the laptop were obviously *constantly* bearing in mind power constraints and were obviously highly innovative and intelligent in designing the most energy efficient laptop that they could, eventually cutting their typical power consumption for one of their laptops to 5-8 Watts—far lower than most other laptops out there. However, in regions where poor/nonexistent power infrastructure is one the primary limiting constraints maybe giving people a laptop (even a very energy efficient one) is not a very wise use of those limited resources. I can't help but notice that the power consumption of the average Raspberry Pi (which aren't even really typically focused a huge amount on reducing energy consumption) is an insignificant fraction of what you see from these laptops whose creators have worked feverishly to reduce power consumption—imagine what could be achieved if they took that same knowledge and skillset and applied it to something lighter and more nimble than a laptop—something a little closer to a raspberry pi. A little Raspberry Pi or Raspberry Pi-esque device like ours without a mouse, keyboard, or monitor plugged into it (all of which would be consuming electricity) and only a cheap little 2x16 or 4x20 standard character lcd screen like the one ours has (whose power consumption is relatively negligible, comparatively), requires only a fraction of the power that one of these laptops would need, at a fraction of the cost. It'd also be significantly lighter, and easier to move/transport. Maybe we need to rethink what makes a computer, a truly viable, functional computer.

We felt that it was important to experiment with many different attempts to wire up a pi using a wide variety of implements, the point of which was to determine the viability of various scavenged materials to perform various jobs related to sending and receiving input and output that would normally require expensive external hardware be purchased and plugged in to perform a given role.

What might be an adequate substitute and what might not? Buttons, resistors, Leds, switches, jumper cables, extension boards/ribbon cables and breadboards and all this various other equipment all cost money—and although these costs might seems small to some, they rapidly begin to accumulate over time. If money and access to specific specialized unique resources are your limiting constraints, then what are some potential ways to overcome said constraints?

Much of this will seem quite superfluous and the truth is, in spite of the fact that I learned a lot in this class, this discussion of makeshift wiring solutions and mistakes is probably the most important thing of all because I learned exactly how far you can push a Raspberry Pi outside of its comfort zone by making use of unconventional wiring solutions without access to proper equipment, and how well those solutions may or not work over the long haul (how well they continue to hold up after they've been in place for hours or days or weeks or months).

I feel compelled to share with you some of the conclusions/solutions we've come to in applying makeshift wiring solutions under these circumstances unusual sorts of circumstances.

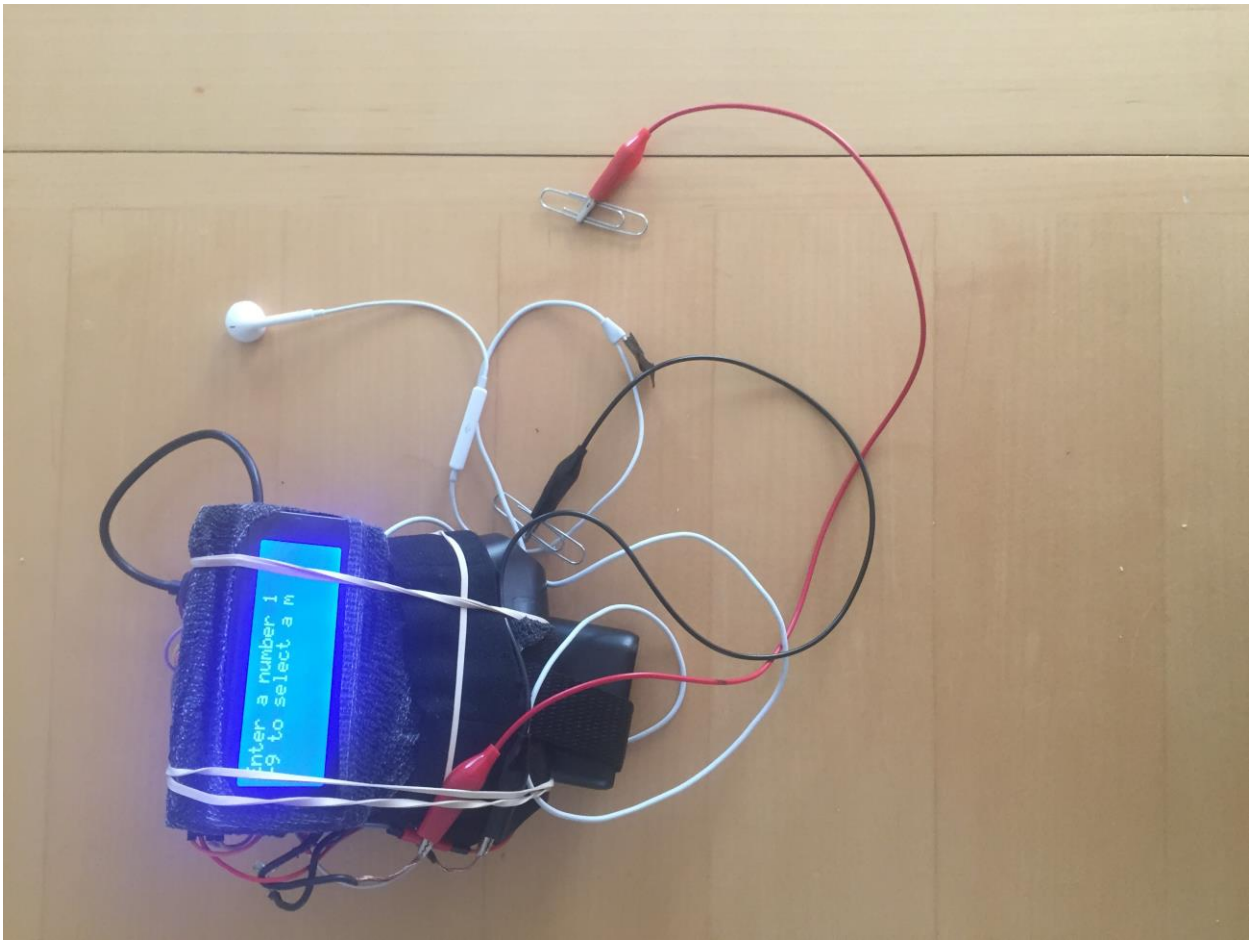


Figure 2. This is the Raspberry Pi we set up with a shell script that would create circumstances such that our python code would execute at the start automatically each time our pi gets turned on. Instead of a button, for THIS raspberry Pi we simply touch two pieces of metal together to perform our button pushing by touching the red and black alligator clips holding paperclips together to press our button. The wiring is otherwise identical.

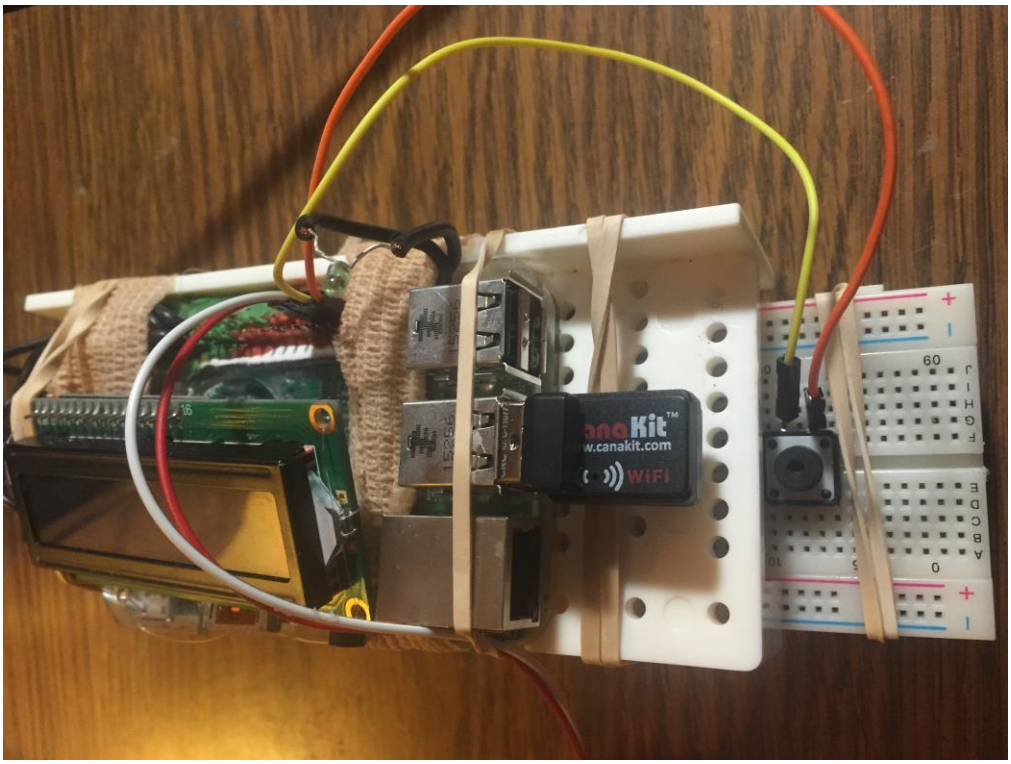
Basically, the limitation of how few resources are available to us under this simulated set of circumstances is actually somewhat helpful in guiding towards a potential solution since there's far less things that needs testing—however, because the number of potential solutions is so small/limited, there's no real reason to not test each and every potential solution (no matter how strange) to a sufficient extent that you can conclude one way or another whether it was a good idea or not-- basically you only need to test out those substances that one might expect to potentially scavenge out of a garbage heap or find lying on the ground.

The first thing I tested was using thin, insulated solid core copper wires whose copper core is roughly the exact same thickness as the GPIO pins on the Raspberry Pi— my idea was that maybe I could use that perfect symmetry to create a perfectly snug fit, thereby a creating a perfect connection that would hold in place perfectly—all that needs to be known about this, is that I tried every single possible variation on making this idea work and have come to the conclusion therefore, that is an incredibly *bad* idea and is not a valid solution under any circumstances.

The next thing I tried was paper clips—it would've been amazing had we gotten such a commonplace item working in connecting up GPIO pins on our Raspberry Pi, but after extremely extensive I conclude with certainty that this is also not a valid solution. However, my partner and I both used paperclips occasionally to connect up various wires, leds and buttons on our breadboard during various iterations of our final project wiring—it works very well for this

purpose if you have no other options and paper clips is all you have (again, I find it far more likely that a person might have a few paperclips on their person than almost anything else which is why I felt that testing this extensively was so important).

The next thing I tried was to pull a small number of strands of thin copper wire out of a thick (sometimes just one) out of a thicker stranded insulated copper wire (each containing ~20-25 very thin strands). I was very nervous about accidentally touching two GPIO pins I wanted to keep separate – this is why I was trying to be so precise. This would never have worked, and it would never have been durable enough to last long enough to be useful even if by some miracle it had. However, counterintuitively, this complete wire (these insulated copper wires of about 20-25 thin strands to each wire) which was far too large/segmented to utilize in a precise manner without risking an accidental short was actually the best possible solution—when applied correctly.



Whenever I read stories about people living in the third world who are able to overcome the sorts of seemingly insurmountable obstacles in order to complete great feats with limited resources, I tend to pay very close attention to the very precise, quantifiable details of every single item that was used during this process—in the case of William Kamkwamba, the impoverished teenager who built a huge electricity-generating wind turbine in the poor slums of Malawi again and again you hear about him utilizing these strips of old discarded PVC pipe that he had to cut out and continually melt and then (while in this highly malleable partially melted state) re-forge—this was done again and again countless times, until finally they came to take on the shape that he wanted. I make this point to drive home that under these sorts of circumstances, finding some way to reuse old trash is not just good idea—it's probably essential, given that it might well be the only resource that you have in abundance (if ever you're in a situation so desperate that you'd actually consider using something like our code as a means of entering/receiving input and output).

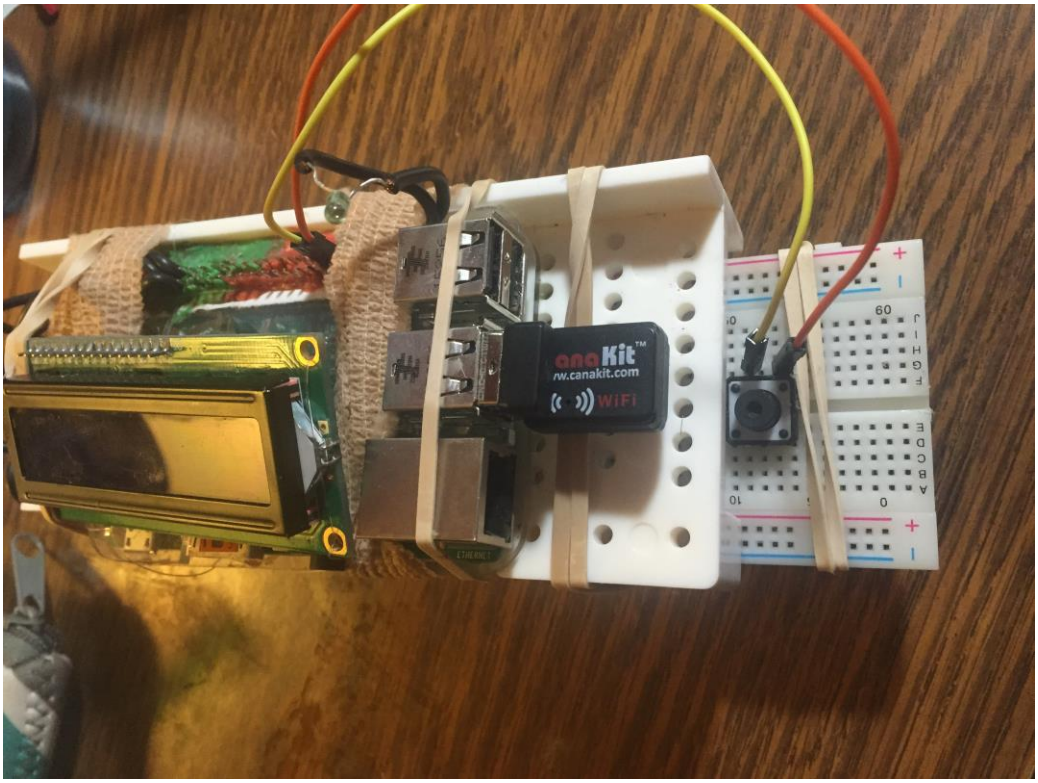


Figure 3.

Rather than go through every single failed attempt at makeshift wiring I attempted over the past few months, I'm just going to jump to the chase the surprisingly effective best possible makeshift wiring method I've discovered so far. I want to stress before I start explaining this, that I have had 2 Pi's assembled using this method almost continuously for every hour for month at least in which time I used it extensively (and was often times fairly rough with my use of it) and essentially had no problem with either pi at any point during that time—in fact the only problems I ever have with our project's wiring comes from when I use jumper cables (which I specifically purchased to hook up to these GPIO pins) instead of these insulated stranded copper wires. I experimented with many different thicknesses and strand numbers but found that this kind of copper wire is the only one (as can be seen in figure 5) that is any way effective/reliable enough to be a viable option.

Rather than list all my experimental mistakes in developing a makeshift wiring technique for hooking up Raspberry Pi GPIO pins without any of the proper/appropriate tools needed to do so, I'll just unapologetically give the entire insane-sounding procedure now—it should be noted that I have only ever used my own Pi's to perform this procedure:

1) Make sure your pi is off throughout this process unless I indicate otherwise. Go get a pack of gum (preferably something sugar-free like trident)
2) Chew a single stick
3) While chewing, cut or rip out a segment of the package of gum (you want a rectangle of just the cardboard/papery material that the gum pack is made out of)—this rectangle should be roughly the size of the long line of GPIO pins on your raspberry pi
4) Slowly press this rectangle into the GPIO gently but firmly until the pins have all completely pierced through and impaled the papery/cardboard material. Continue pressing until the cardboard-like material

is flush with whatever you have underneath. This is mostly done to protect the pi from any potential risk of damage, but also helps to secure everything into place

5) Using the entire piece of gum you just chewed, apply a tiny amount of gum (in the form of 6 approximately equal sized "threads" of chewing gum, about an inch long) all around the GPIO pins where you're likely to plug your makeshift wires, making using a pen/pencil tip or any random implement that's thin enough to that the chewing gum is pressed relatively flush (it's not exact, just enough to that it's secured in place).
6) Now place your stranded copper wires in the following way:
    a. Cut out a bunch of stranded copper wire segments using scissors, wire cutters, or any random implement sharp enough to go through
        i. Any random assortment of wire lengths approximately 2-9 inches long each
    b. Gently but firmly press the truncated cut head of one of the wires into the appropriate GPIO pin, and the other end into its appropriate companion. If it gives you any trouble inserting it in this manner, don't force it—instead for anything like that, just insert it such that the rubber insulated of the wire is hugging/squeezing the GPIO pin (pressing against it directly), thereby holding it tightly and securely to the copper strands. The connection tends to be slightly more secure however when the GPIO pin gets shoved directly into the *center* of the stranded copper wire
    c. At this point you should then grab the part of the wire that's plugged into this pin and kink it at a 90 degree angle like you're kinking a hose. Then try to make friction your friend as often as you can here—**we really don't want any of these wires jostling around unnecessarily** therefore use one of the following methods to secure our wire effectively using **friction**:
        i. Strap a mesh/grid-like structure onto your pi to act as a base to secure things to, then weave your wire once or twice through one or two of the holds in this mesh/grid.
        ii. If lazy just try to keep the wire pressed close to solid surfaces—any little bit of friction to hold things in place will help
7) Now you want to take a piece of breathable athletic tape (the kind that only sticks to itself rather than being sticky) and select a place to begin binding down the wire (just tight enough to keep them from jostling). Before doing so, if there's any extra length on any of your wires, instead of having these parts of the wiring sticking out, have these wires simply double back on themselves a couple times (making an "S"

or “W” shape) right around the wires you’re going to bind down with athletic tape.

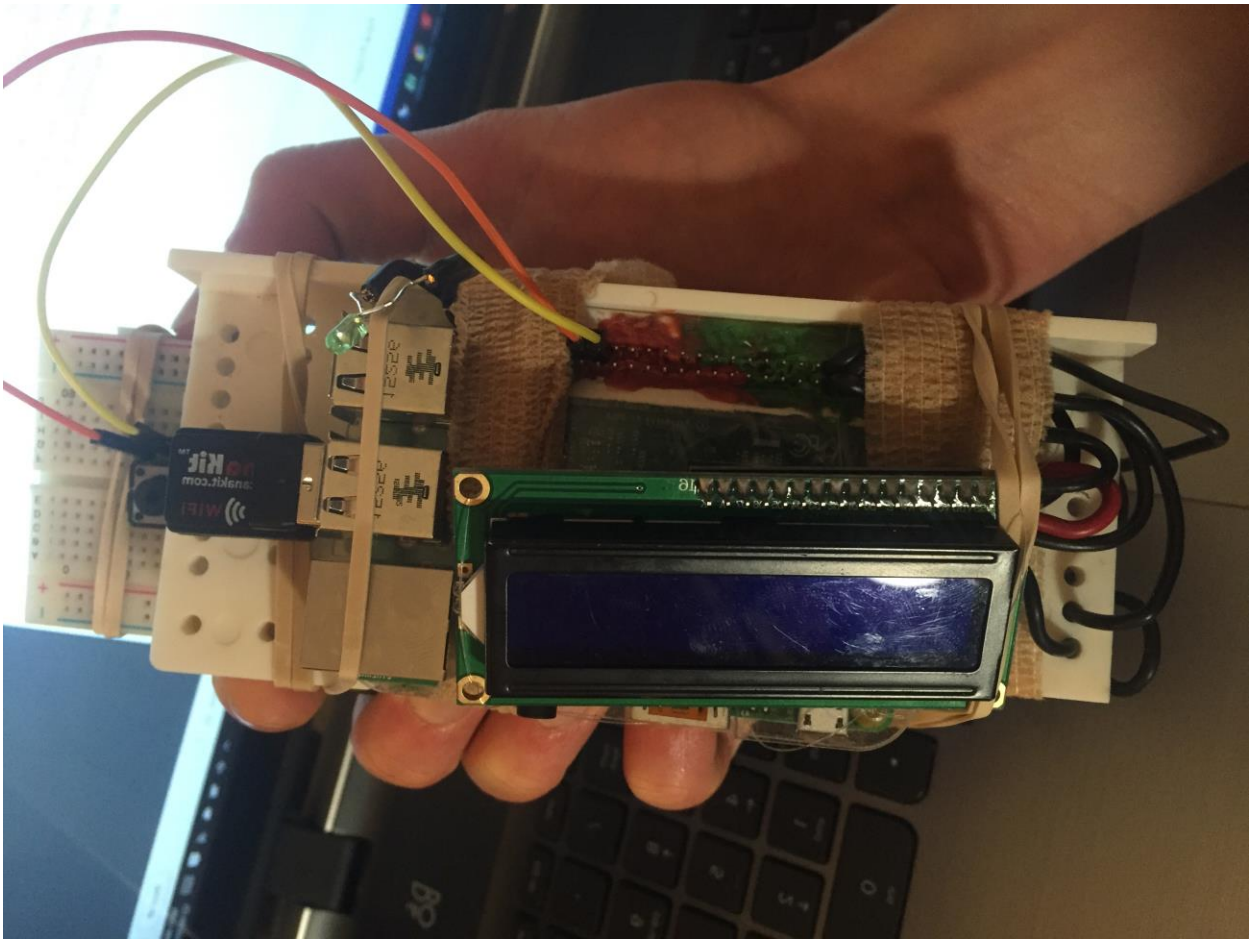


Figure 4. Here you see a clear example of how we made use of existing items like this Miter board which happens to already have holes in it that are approximately the same diameter as our copper stranded wire allowing us to more easily secure our wires by weaving them through the Miter Board holes in order to increase friction and decrease wire jostling

8) After this is done, add a rubber band or two on top of the athletic tape to make sure that it is all completely secured and isn’t likely to just randomly come loose at some point by accident
9) As a final touch, any place where multiple copper wires are getting hooked up to multiple pins very close to one another, it creates a slight amount of risk that things might accidentally connect up in a way that we don’t want (since our stranded copper wires our usually ever so slightly frayed and splaying out at the base. If ever you feel that too much of the copper strands are peeking out, just trim a quarter centimeter or so off the end of the wire to clean it up. In addition, using a slight amount of chewing gum around these areas to insulate them a little bit (thereby reducing the risk of accidental collision) in this area as well. Although using chewing gum in this manner might seem very strange, the reason why I do this is that it is completely obvious from my testing that *not* adding chewing gum (after attaching copper stranded wire to our pins) will result in a slight possibility that every time you touch your pi or move it around or brush against it or jostle it in exactly the wrong way there’s a slight possibility the entire pi might unexpectedly crash and reset, causing you to completely destroy all of your progress on whatever you’re currently working on since your last save. Obviously, even a *slight* possibility that this could happen could be catastrophic at the wrong time—therefore, anything that can be done that would significantly reduce this risk is worthwhile.

The only absolutely essential aspects to this method of applying this makeshift wiring is that you currently possess a kind of copper stranded wire sufficiently similar to that which I have described above (and is shown below in figure 2) and at least one piece of chewed chewing gum.

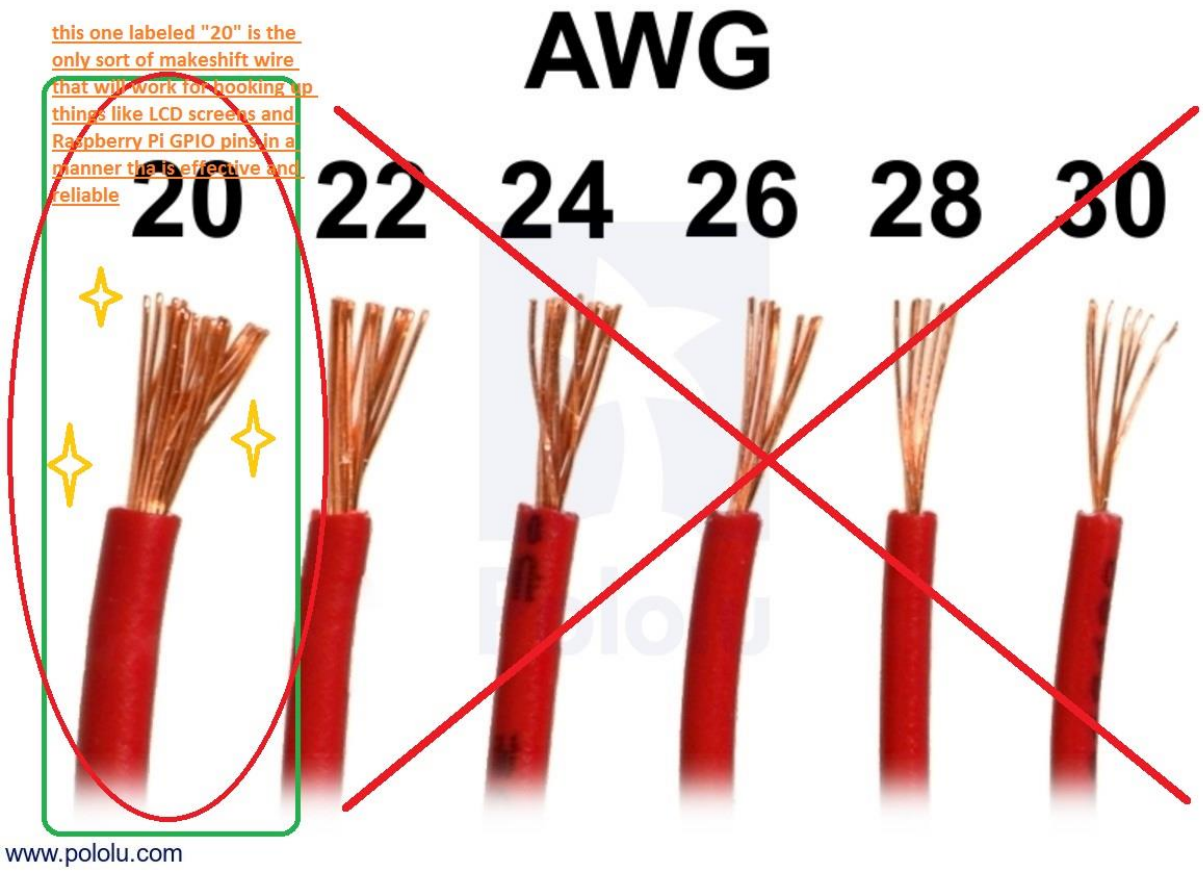


Figure 5. This demonstrate the sorts of makeshift wire types that would NOT work --compared to the one on the far left-- which does.

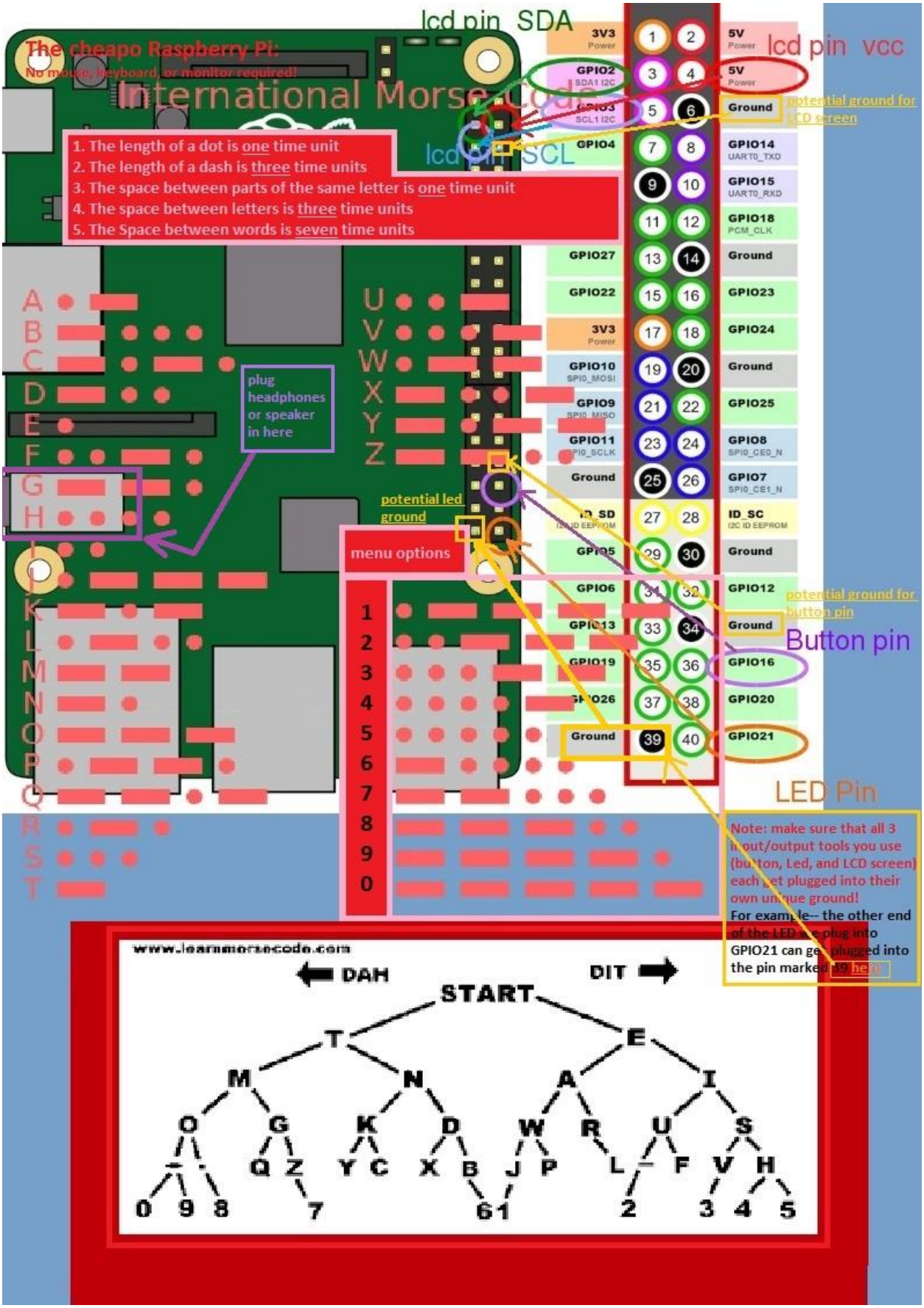

lcd pin SDA
lcd pin vcc
The cheapo Raspberry Pi:
No mouse, keyboard, or monitor required!
International Morse Code
1. The length of a dot is one time unit
2. The length of a dash is three time units
3. The space between parts of the same letter is one time unit
4. The space between letters is three time units
5. The Space between words is seven time units
lcd pin SCL
potential ground for LCD screen
plug headphones or speaker in here
potential led ground
menu options
1 2 3 4 5 6 7 8 9 0
potential ground for button pin
Button pin
LED Pin
Note: make sure that all 3 input/output tools you use (button, Led, and LCD screen) each get plugged into their own unique ground!
For example-- the other end of the LED can plug into GPIO21 can get plugged into the pin marked 39 here
www.learnmorsecode.com
DAH
DIT
START
T E
M N A I
O G K D W R U S
Q Z Y C X B J P L F V H
0 9 8 7 61 2 3 4 5

Figure 6. Raspberry Pi diagram with Morse Code inputs and outputs we used. This shows slightly more detailed wiring than the previous diagram

The Code:

```
import os
import RPi.GPIO as GPIO
import time
# bare minimum terminal commands
import random

import urllib.request

import subprocess
#lcdglo: lines=0, charPerLine=1, timeUnit=2, textDisplaySpeed=3, ledPin = 4, blinkOutputAi(on/off state)=5, subDirectory=6, buttonPin=7
lcdGlo = [(2),(16),(0.14),(0.35),(21),(0),(""),(16)]
#lcdGlo[0]=int(lcdGlo[0])
#numCharLcdSet=int(lcdGlo[1])
print("num of lines: ", lcdGlo[0])
print("num of char per line: ", lcdGlo[1])

ButtonPin=lcdGlo[7]
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(lcdGlo[4], GPIO.OUT)
GPIO.output(lcdGlo[4], GPIO.LOW)

GPIO.setup(ButtonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)

createdFileName=""


print("time unit value: ", lcdGlo[2])

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(lcdGlo[4], GPIO.OUT)#set LED pin's mode as output
GPIO.output(lcdGlo[4], GPIO.LOW)#sets LED pin to be OFF at start

GPIO.setup(ButtonPin, GPIO.IN, pull_up_down = GPIO.PUD_UP)#this sets our ButtonPin's mode as input with the internal pullup

```

```
#bare minimum LCD screen:
#LCD pins assigned are VERY simple-- just follow this:
# vcc goes to 5 volts
# GND goes to some ground (either the long blue line marked with the negative sign or any pin marked with "GND")
# SDA goes to SDA (or SDA1 if you've got it)
# SCL goes to SCL (or SCL1 if that's what you see)

import datetime

# Modified 2018-09-30 to use Matt Hawkins' LCD library for I2C available
# from https://bitbucket.org/MattHawkinsUK/rpispy-misc/raw/master/python/lcd_i2c.py
import RPi.GPIO as GPIO
import time

# Additional stuff for LCD
import smbus

xGlo= [(0x27),(0x80),(0xC0),(0x94),(0xD4)]
print("I2C device address: ", xGlo[0])
print("ram address line 1: ", xGlo[1])
print("ram address line 2: ", xGlo[2])
print("ram address line 3: ", xGlo[3])
print("ram address line 4: ", xGlo[4])


#LCD pin assignments, constants, etc
I2C_ADDR  = 0x27 # I2C device address
#int(lcdGlo[1]) = lcdGlo[1]   # Maximum characters per line

# Define some device constants
LCD_CHR = 1 # Mode - Sending data
LCD_CMD = 0 # Mode - Sending command

LCD_LINE_1 = 0x80 # LCD RAM address for the 1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the 2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the 3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the 4th line

LCD_BACKLIGHT  = 0x08  # On
#LCD_BACKLIGHT = 0x00  # Off

```

```
ENABLE = 0b00000100 # Enable it

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005


# Ultrasonic pin assignments
SR04_trigger_pin = 20
SR04_echo_pin = 21

# LCD commands
LCD_CMD_4BIT_MODE = 0x28   # 4 bit mode, 2 lines, 5x8 font
LCD_CMD_CLEAR = 0x01
LCD_CMD_HOME = 0x02   # goes to position 0 in line 0
LCD_CMD_POSITION = 0x80  # Add this to DDRAM address

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)       # Numbers GPIOs by standard marking

# Set up the SR04 pins
GPIO.setup(SR04_trigger_pin, GPIO.OUT)
GPIO.setup(SR04_echo_pin, GPIO.IN)
GPIO.output(SR04_trigger_pin, GPIO.LOW)

#Open I2C interface
#bus = smbus.SMBus(0)  # Rev 1 Pi uses 0
bus = smbus.SMBus(1) # Rev 2 Pi uses 1


def oneBookendGrab(textToCutSingle,startSign):
    bookendSingle=textToCutSingle.find(startSign)
    return (textToCutSingle[bookendSingle+(len(startSign)):])

def PlugTrackIntoArray(textToCutSingle,endSign):
    bookendSingle=textToCutSingle.find(endSign)
    return (textToCutSingle[:bookendSingle])

def finalTrackExtract(endOfTrack1,musicTrack1):

    finalTrack1=""
```

```
    headsUpSign1=""

    headsUpSign1="<p>"
    if(endOfTrack1.count("Text")>0):
        headsUpSign1="Text\":\""

    numSongsLeft=musicTrack.count(headsUpSign1)
    musicTrackArray1=""
    musicTrackArray2=[]
    countForRef=0
    while (numSongsLeft>1):
        numSongsLeft=numSongsLeft-1
        print("Paragraphs left:", numSongsLeft)
        musicTrack1=oneBookendGrab(musicTrack1,headsUpSign1)

        finalTrack1=PlugTrackIntoArray(musicTrack1,endOfTrack1)
        print(finalTrack1)
        countForRef=countForRef+1
        if (musicTrack1.count("duckduckgo")>=1 and countForRef==1):
            lcdSays("duckduckgo search engine results displaying: ")
        else:
            musicTrackArray1=(str(musicTrackArray1)+"  Paragraph "+str(countForRef)+": "+str(finalTrack1))
        if (numSongsLeft > 7 and countForRef>3):
            return musicTrackArray1
        print(musicTrackArray1)
    return musicTrackArray1

def lcd_init():
  try:
    lcd_byte(0x33,LCD_CMD) # 110011 Initialise
    lcd_byte(0x32,LCD_CMD) # 110010 Initialise
    lcd_byte(0x06,LCD_CMD) # 000110 Cursor move direction
    lcd_byte(0x0C,LCD_CMD) # 001100 Display On,Cursor Off, Blink Off
    lcd_byte(0x28,LCD_CMD) # 101000 Data length, number of lines, font size
    lcd_byte(0x01,LCD_CMD) # 000001 Clear display
    time.sleep(E_DELAY)
  except:
    print("LCD screen not currently plugged in")

def lcd_byte(bits, mode):
  bits_high = mode | (bits & 0xF0) | LCD_BACKLIGHT
  bits_low = mode | ((bits<<4) & 0xF0) | LCD_BACKLIGHT
```

```
  # High bits
  bus.write_byte(I2C_ADDR, bits_high)
  lcd_toggle_enable(bits_high)

  # Low bits
  bus.write_byte(I2C_ADDR, bits_low)
  lcd_toggle_enable(bits_low)

def lcd_toggle_enable(bits):
  # Toggle enable
  time.sleep(E_DELAY)
  bus.write_byte(I2C_ADDR, (bits | ENABLE))
  time.sleep(E_PULSE)
  bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
  time.sleep(E_DELAY)

def lcd_string(message,line):
  # Send string to display
  try:
    message = message.ljust(int(lcdGlo[1])," ")

    lcd_byte(line, LCD_CMD)

    for i in range(int(lcdGlo[1])):
      lcd_byte(ord(message[i]),LCD_CHR)
  except:
    print(message)

# functions not in the original library
def lcd_xy(col, row):
      lcd_byte(LCD_CMD_POSITION+col+(64*row), LCD_CMD)

def lcd_msg(msg_string):
      for i in range(0, len(msg_string)):
            lcd_byte(ord(msg_string[i]), LCD_CHR)


def readFromFile(file_in):
    """
    Counts the number of lines in a file_in
    file_in: string - file name
```

```
    Returns: integer - number of lines in file_in
    """

    # create file object fin for reading

    fin = open(file_in,'r')

    # create and initialize local variable to store result
    numLines = 0


    # iterate through fin with line loop variable
    for line in fin:
        numLines = numLines + 1
        #print(line)
        lcdSays(line)


    # close fin object
    fin.close()

    # return result
    return line

#/////////////////////////////////////////////////////////////////////////

def readFromFileArr(file_in):
    """
    Counts the number of lines in a file_in
    file_in: string - file name
    Returns: integer - number of lines in file_in
    """

    # create file object fin for reading

    fin = open(file_in,'r')

    # create and initialize local variable to store result
    numLines = 0

    arrSongs=[""]
    arrSongs
```

```
    # iterate through fin with line loop variable
    for line in fin:

        numLines = numLines + 1
        print("line is", line)
        arrSongs.append(line)
        print("Array Song number",numLines,"is", arrSongs[numLines])

    # close fin object
    fin.close()

    # return result
    return arrSongs

#////////////////////////////////////////////////////////////////////////

def quickGrabAll(inputUrl,outputText):
    grabBytes= urllib.request.urlopen(inputUrl)
    bText = grabBytes.read()
    outputText = bText.decode("utf8")
    grabBytes.close()
    return outputText


# `O+:+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX///:
# /+.`./XXXOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO-.`.X
# /X/:+XXXXXXXXOOOOOOOOOOOOOOOOOOOOOOOOOOOO++XO
# `-```/OOOOOOOOXXXOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[3V3][5V]
# `-```/OOOOOOOOOOOOOOOO+:+-XOXO+OOOOOOO(@)(@)Z---->[SDA1 (SDA)][5V (vcc)]<---LCD SCREEN
# `-```/OOOOOOOOOOOOOOO-O/+OX::XX/OOOOOOO(@)(@)Z---->[SCL1 (SCL)][Ground (gnd)]<---LCD SCREEN
# /OOOOOOOOOOOOOOOOO+/OX/O++:-OOOOOOOOO(@)(@)Z---->[GPIO4][GPIO14]
# /OOOOOOOOOOOOOOOOO/XO:X///XX/OOOOOOO(@)(@)Z---->[Ground][GPIO15]
# /OOOOOOOOOOOOOOOOOOOOOOOOXXOXXXOOOOOOO(@)(@)Z---->[GPIO17][GPIO18]
# .+XXXXXXXXOOOOOOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[GPIO27][Ground]
# :```````:OOOOOOOOOO+++++++++XOOOOOOOO(@)(@)Z---->[GPIO22][GPIO23]
# :```````:OOOOOOOOOO+++++++++XOOOOOOOO(@)(@)Z---->[
# :```````:OOOOOOOOOO+++++++++XOOOOOOOO(@)(@)Z---->[
# :```````:OOOOOOOOOO+++++++++XOOOOOOOO(@)(@)Z---->[
# :```````:OOOOOOOOOOOOOXXXXXXOOOOOOOOO(@)(@)Z---->[
# ./XXXXXXXXOOOOOOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[
# /OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[
```

```
#  /OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[GPIO5][Ground]
#  /OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[GPIO6][GPIO12]
#  /XXXXXXXXXXOOOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[GPIO13][Ground]
#`OO.```````/OOOOOOOOOOOOOOOOOOOOOOOO(@)(@)Z---->[GPIO19][GPIO16 (button pin)]<
#`OO.```````/OOOOOOOOOOOOOO++++++OOOOOOO(@)(@)Z---->[GPIO26][GPIO20]
# .+OOOOOOOOOOOOOOOOOOOOOOOO++++++OOOOOOO(@)(@)Z---->[GPIO21 (Led pin)][Ground]<
#  /X:-:XOOOOOOOOOOOOOOOOOO++++++OOOOOOOOO:-/X
#  /O.`.OOOOOOOOOOOOOOOOOOOOOOXXXOOOOOOOOO:.`.X
#  /OXXXXXXXXXXXXXXXOOOOOOOOOOOOOOOOOOOOOOOOOOOO
#  /OO-``````````OOOOOOOOOOOOOOOOOOOOOOOOOOOO
#  /OO-``````````OOOOOOOOOOOOOOOOOOOOOOOOOOOO
#  /OO-``````````OOOO````````-OOOO`````````OO
#  /OO-``````````OOOO````````-OOOO`````````OO
#  /OO-``````````OOOO````````-OOOO`````````OO
#  /OO-``````````OOOO````````-OOOO`````````OO
#  /OO-``````````OOOO````````-OOOO`````````OO
#   :/-``````````////````````-+//+````````.O-
#    .`````````` #`````````  #`````````
```

```
#now we define the morse code function:
#This conversion would be about converting dots/dashes to letters)
def dd_to_let(convertedLetter):
    if(convertedLetter=='.-'):
        convertedLetter='a'
    elif(convertedLetter=='-...'):
        convertedLetter='b'
    elif(convertedLetter=='-.-.'):
        convertedLetter='c'
    elif(convertedLetter=='-..'):
        convertedLetter='d'
    elif(convertedLetter=='.'):
        convertedLetter='e'
    elif(convertedLetter=='..-.'):
        convertedLetter='f'
    elif(convertedLetter=='--.'):
        convertedLetter='g'
    elif(convertedLetter=='....'):
        convertedLetter='h'
    elif(convertedLetter=='..'):
        convertedLetter='i'
    elif(convertedLetter=='.---'):
        convertedLetter='j'
```

```
elif(convertedLetter=='-.-'):
    convertedLetter='k'
elif(convertedLetter=='.-..'):
    convertedLetter='l'
elif(convertedLetter=='--'):
    convertedLetter='m'
elif(convertedLetter=='-.'):
    convertedLetter='n'
elif(convertedLetter=='---'):
    convertedLetter='o'
elif(convertedLetter=='.--.'):
    convertedLetter='p'
elif(convertedLetter=='--.-'):
    convertedLetter='q'
elif(convertedLetter=='.-.'):
    convertedLetter='r'
elif(convertedLetter=='...'):
    convertedLetter='s'
elif(convertedLetter=='-'):
    convertedLetter='t'
elif(convertedLetter=='..-'):
    convertedLetter='u'
elif(convertedLetter=='...-'):
    convertedLetter='v'
elif(convertedLetter=='.--'):
    convertedLetter='w'
elif(convertedLetter=='-..-'):
    convertedLetter='x'
elif(convertedLetter=='-.--'):
    convertedLetter='y'
elif(convertedLetter=='--..'):
    convertedLetter='z'
elif(convertedLetter=='.----'):
    convertedLetter='1'
elif(convertedLetter=='..---'):
    convertedLetter='2'
elif(convertedLetter=='...--'):
    convertedLetter='-'
elif(convertedLetter=='....-'):
    convertedLetter='3'
elif(convertedLetter=='....-'):
    convertedLetter='4'
```

```
    elif(convertedLetter=='.....'):
        convertedLetter='5'
    elif(convertedLetter=='-....'):
        convertedLetter='6'
    elif(convertedLetter=='--...'):
        convertedLetter='7'
    elif(convertedLetter=='---..'):
        convertedLetter='8'
    elif(convertedLetter=='----.'):
        convertedLetter='9'
    elif(convertedLetter=='-----'):
        convertedLetter='0'
    elif(convertedLetter=='----'):
        convertedLetter=' '
    elif(convertedLetter=='---.'):
        convertedLetter='.'
    else:
        convertedLetter='?'
    return convertedLetter

#-------------------------------------------------------------
def end_message(convertedMessage):
    convertedMessage=(str(convertedMessage)+'. ---OUT---')
    print('COMPLETE MESSAGE: ', str(convertedMessage))
    return convertedMessage
#-------------------------------------------------------------


def run_new_letter_funcs(ddCombo, convertedLetter):
    convertedLetter=ddCombo
    #print('Debug: our midconversion letter is ', str(convertedLetter))
    convertedLetter=dd_to_let(convertedLetter)
    print('our new letter is ', str(convertedLetter))
    return convertedLetter


#-------------------------------------------------------------
def add_word_to_message(convertedWord,convertedMessage):
    convertedMessage=(str(convertedMessage)+' '+str(convertedWord))
    print('our current message so far is ', str(convertedMessage))
    return convertedMessage
#-------------------------------------------------------------
```

```
def add_letter_to_word(convertedLetter,convertedWord):
    convertedWord=(str(convertedWord)+str(convertedLetter))
    print('our current word so far is ', str(convertedWord))
    return convertedWord

#------------------------------------------------------------


def specifictionary(filterToApply, messageToFilter):
  filteredMessage=""
  if (filterToApply == "Display Settings"):
    filteredMessage=messageToFilter
    filteredMessageU=filteredMessage.upper()
    if (filteredMessageU.count("INCH")==0 and filteredMessageU.count("3.5")==0 and filteredMessage[0].upper() != "3"):
      if (filteredMessageU.count("LCD")>=1 or filteredMessageU.count("4")>=1 or filteredMessageU.count("LINE")>=1 or
filteredMessageU.count("CHAR")>=1 or filteredMessageU.count("I2C")>=1):
        filteredMessage
        if (filteredMessage.upper() == "4" or filteredMessage.upper() == "I2C LCD DISPLAY"):
          print("You selected:")
          print("4- Change number of lines and characters per line displayed by I2C LCD display")
          filteredMessage="I2C LCD Display"
        else:
          filteredMessage="I2C LCD Display"
          print("We think you meant: "+str(filteredMessage))
      return filteredMessage


def writeToFile(createdFileNameB,userSays1,inputModeW):
  if(userSays1==""):
    if (inputModeW=="Keyboard"):
      userSays1=input("What text would you like to enter into the file?")
    else:
      userSays1=morseCode(userSays1,"")
  with open('/home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(createdFileNameB), 'w') as f:
      f.write(""+userSays1)

def writeSettingsToFile(createdFileNameC,userSays1,inputModeW):
  if(userSays1==""):
    if (inputModeW=="Keyboard"):
      userSays1=input("What text would you like to enter into the file?")
    else:
```

```
        userSays1=morseCode(userSays1,"")
    with open('/home/pi/pyAtStartBman/'+str(createdFileNameC), 'w') as f:
        f.write(""+userSays1)


def lcdSays(lcdTextIn):

    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(lcdGlo[4], GPIO.OUT)
    GPIO.output(lcdGlo[4], GPIO.LOW)

    global xGlo
    global lcdGlo


    line1Arr=0*lcdGlo[1]
    line2Arr=1*lcdGlo[1]
    line3Arr=3*lcdGlo[1]
    line4Arr=4*lcdGlo[1]

    line1S=""
    line2S=""
    countLcdArr=0
    totalLinesToPrint=len(lcdTextIn)

    print(lcdTextIn)

    ButtonPin=lcdGlo[7]


    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(ButtonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)



    while(line2Arr<(totalLinesToPrint)):
```

```
        time.sleep(lcdGlo[3])
        line1S= lcdTextIn[line1Arr:(line1Arr+lcdGlo[1])]
        #print(line1S)
        lcd_string(""+str(line1S)+"          ", xGlo[1])

        line2S= lcdTextIn[line2Arr:(line2Arr+lcdGlo[1])]
        #print(line2S)
        lcd_string(""+str(line2S)+"          ", xGlo[2])
        time.sleep(lcdGlo[3])
        #print(""+str(line1S)+str(line2S))

        if(0 == GPIO.input(ButtonPin)):
            print(lcdTextIn)
            try:
                lcd_string("Skipping message...                ", LCD_LINE_1)
                lcd_string("Skipping message...                ", LCD_LINE_2)
            except:
                print("lcd screen is not correctly plugged in")
            time.sleep(lcdGlo[3])
            time.sleep(lcdGlo[3])
            return

        if(lcdGlo[0]>=4):
            time.sleep(lcdGlo[3])
            line1S= lcdTextIn[line1Arr:(line1Arr+lcdGlo[1])]
            #print(line1S)
            lcd_string(""+str(line1S)+"          ", xGlo[3])

            line2S= lcdTextIn[line2Arr:(line2Arr+lcdGlo[1])]
            #print(line2S)
            lcd_string(""+str(line2S)+"          ", xGlo[4])
            time.sleep(lcdGlo[3])

            line1Arr=line1Arr+lcdGlo[1]
            line2Arr=line2Arr+lcdGlo[1]



        if(0 == GPIO.input(ButtonPin)):
            print(lcdTextIn)
            try:
                lcd_string("Skipping message...                ", LCD_LINE_1)
```

```
            lcd_string("Skipping message...                    ", LCD_LINE_2)
        except:
            print("lcd screen is not correctly plugged in")
        time.sleep(lcdGlo[3])
        time.sleep(lcdGlo[3])
        return

    if(lcdGlo[5]==1):
        d2=[]
        l=(""+str(line1S)+str(line2S))
        #l = input("Now how about super profound?  ")
        for i in range(0, len(l)):
            #print(l[i]," in Morse is ",MorseConvert(l[i]), \
                #" which looked up again is ",MorseConvert(MorseConvert(l[i])))
            d2.append(MorseConvert(l[i]))
            if(0 == GPIO.input(ButtonPin)):
                print(l)
                print(d2)
                print(lcdTextIn)
                try:
                    lcd_string("Skipping message...                    ", LCD_LINE_1)
                    lcd_string("Skipping message...                    ", LCD_LINE_2)
                except:
                    print("lcd screen is not correctly plugged in")
                time.sleep(lcdGlo[3])
                time.sleep(lcdGlo[3])
                return
        AiLedBlinkTimes=[]
        for i in range(0, len(d2)):
            for j in range(0, len(d2[i])):
                #print("debug print out our dots and dashes: ", d2[i][j])
                if(d2[i][j] == "."):
                    AiLedBlinkTimes.append(1)
                if(d2[i][j] == "-"):
                    AiLedBlinkTimes.append(3)
                if(d2[i][j] == " "):
                    AiLedBlinkTimes.append(-2)
                if(d2[i][j] == "&"):
                    AiLedBlinkTimes.append(-7)
                if(d2[i][j] == "?"):
                    AiLedBlinkTimes.append(-7)
```

```
            if(0 == GPIO.input(ButtonPin)):
                print(l)
                print(d2)
                print(lcdTextIn)
                try:
                    lcd_string("Skipping message...                ", LCD_LINE_1)
                    lcd_string("Skipping message...                ", LCD_LINE_2)
                except:
                    print("lcd screen is not correctly plugged in")
                time.sleep(lcdGlo[3])
                time.sleep(lcdGlo[3])
                return


    for i in range(0, len(AiLedBlinkTimes)):
        #print(AiLedBlinkTimes[i])
        if(AiLedBlinkTimes[i]>0):
            #print("Led ON")
            GPIO.output(lcdGlo[4], GPIO.HIGH)
            #print((AiLedBlinkTimes[i])*lcdGlo[2])

            time.sleep((AiLedBlinkTimes[i])*lcdGlo[2])
            #print("Led OFF")
            GPIO.output(lcdGlo[4], GPIO.LOW)
            time.sleep(1*lcdGlo[2])
            #print(str(1*lcdGlo[2]))
        else:
            #print("Led OFF")
            GPIO.output(lcdGlo[4], GPIO.LOW)
            time.sleep(-1*lcdGlo[2]*(AiLedBlinkTimes[i]))
            #print(-1*lcdGlo[2]*(AiLedBlinkTimes[i]))

            if(0 == GPIO.input(ButtonPin)):
                print(l)
                print(d2)
                print(lcdTextIn)
                try:
                    lcd_string("Skipping message...                ", LCD_LINE_1)
                    lcd_string("Skipping message...                ", LCD_LINE_2)
                except:
                    print("lcd screen is not correctly plugged in")
                time.sleep(lcdGlo[3])
```

```
                time.sleep(lcdGlo[3])
                return
        line1Arr=line1Arr+lcdGlo[1]
        line2Arr=line2Arr+lcdGlo[1]









    if (0 == GPIO.input(ButtonPin)):

        GPIO.output(lcdGlo[4], GPIO.HIGH)
        print("Message  WITHOUT LED blinks in Morse Code")
        time.sleep(1)

        GPIO.output(lcdGlo[4], GPIO.LOW)



    while(line2Arr<(totalLinesToPrint) and 0 == GPIO.input(ButtonPin)):

        time.sleep(lcdGlo[3])
        line1S= lcdTextIn[line1Arr:(line1Arr+lcdGlo[1])]
        #print(line1S)
        lcd_string(""+str(line1S)+"          ", xGlo[1])

        line2S= lcdTextIn[line2Arr:(line2Arr+lcdGlo[1])]
        #print(line2S)
        lcd_string(""+str(line2S)+"          ", xGlo[2])
        time.sleep(lcdGlo[3])
        print(""+str(line1S)+str(line2S))

        line1Arr=line1Arr+lcdGlo[1]
        line2Arr=line2Arr+lcdGlo[1]

        if(lcdGlo[0]>=4):
            time.sleep(lcdGlo[3])
            line1S= lcdTextIn[line1Arr:(line1Arr+lcdGlo[1])]
            #print(line1S)
            lcd_string(""+str(line1S)+"          ", xGlo[3])
```

```
        line2S= lcdTextIn[line2Arr:(line2Arr+lcdGlo[1])]
        #print(line2S)
        lcd_string(""+str(line2S)+"          ", xGlo[4])
        time.sleep(lcdGlo[3])

        line1Arr=line1Arr+lcdGlo[1]
        line2Arr=line2Arr+lcdGlo[1]









#=============================================================================================================
==
Morse2 = ['a','.- ','b','-... ','c','-.-. ','d','-.. ','e','. ','f','..-. ','g','--. ', \
    'h','.... ','i','.. ','j','.--- ','k','-.- ','l','.-.. ','m','-- ','n','-. ','o','--- ','.','---.', \
    'p','.--. ','q','--.- ','r','.-. ','s','... ','t','- ','u','..- ','v','...- ','-','--..', \
    'w','.-- ','x','-..- ','y','-.-- ','z','--.. ','1','.---- ','2','..--- ','3','...-- ', \
    '4','....- ','5','..... ','6','-.... ','7','--... ','8','---.. ','9','----. ','0','----- ',' ','&']


#=============================================================================================================
==
#=============================================================================================================
==
def MorseConvert(s):
    global Morse2
    try:
        if s[0] < '0':
            retval = Morse2[Morse2.index(s)-1]
        else:
            retval = Morse2[Morse2.index(s[0].lower())+1]
    except:
            retval = '?'
    return retval
#=============================================================================================================
==

def email_sending(receiver,content):


```

```
#email_sending('brentxavierproject@gmail.com','blah bhlafdsafdsa')


def checkEmail():

  import poplib

  from email import parser

  pop_conn = poplib.POP3_SSL('pop.gmail.com')

  pop_conn.user('brentxavierproject@gmail.com')

  pop_conn.pass_('Pythonpass123-')

```

```
    #pulls the content from the server
    messages = [pop_conn.retr(i) for i in range(1, len(pop_conn.list()[1]) + 1)]

    # puts back together messages for printing
    #messages = ["\n".join(mssg[1]) for mssg in messages]
    #print (messages)
    #print("Hello world")
    messages2 = ""
    mcount=0
    for mssg in messages:
        mcount=mcount+1
        #print("Message ",mcount," = ", str(mssg))
        #print (mssg)
        #messages2 = messages2+str(mssg)+"\r\n\r\n"
        messages2 = str(mssg)+"\r\n\r\n"

        #print(messages2)
        #print("")
        search1=messages2.find("b'From:")
        search2 = messages2.find("b'", search1+1)
        search3 = messages2.find("b'",search2+1)
        search4 = messages2.find("']", search3+1)
        #print(subjloc,subjend,msgstart, msgstart2,msgend)
        print(messages2[search3+2:search4])
        lcdSays("The following email messages were found: "+str((messages2[search3+2:search4])))
    emailDisplay1=""
    #print("There were ",mcount," messages")
    emailDisplay1=("There were ",mcount," messages")
    lcdSays(emailDisplay1)

    pop_conn.quit()



def morseCode(textInOut, specFilter):


    #lcdGlo[2]=0.14

    #lcdGlo[4] = 21
```

```
    ButtonPin=lcdGlo[7]

    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(lcdGlo[4], GPIO.OUT)
    GPIO.output(lcdGlo[4], GPIO.LOW)

    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(lcdGlo[4], GPIO.OUT)#set LED pin's mode as output
    GPIO.output(lcdGlo[4], GPIO.LOW)#sets LED pin to be OFF at start

    GPIO.setup(ButtonPin, GPIO.IN, pull_up_down = GPIO.PUD_UP)#this sets our ButtonPin's mode as input with the
internal pullup

    OFFtimer=0 #this is the amount of time that the LED was OFF for
    ONtimer=0 #this is the amount of time that the LED was ON for

    ONtimerStarted=0 #this checks if our ONtimer has started ticking yet-- if is HAS,
    # then we we set ONtimerStarted=1... when our timer turns off, ONtimerStarted is
    # reset back to ZERO.  OFFtimerStarted follows the same basic principle.
    OFFtimerStarted=0

    startTimeOn=1.111 #records exact time at the moment the LED is turned ON
    startTimeOff=1.111 #records exact time at the moment the LED is turned OFF

    onList = []
    offList = []

    numButtonPushes = 0
    numButtonReleases = 0

    waitTime=0

    input_state = GPIO.input(ButtonPin)

    #==============SECTION 2: DEFINING FUNCTIONS (AND DECLARING VARIABLES THAT USE THEM)========
```

```
#=============================================================================================

convertedLetter='c'
convertedWord=''
convertedMessage='Message: '

time.sleep(lcdGlo[3])
try:
  lcd_string("Enter Morse Code:                ", LCD_LINE_1)
  lcd_string("                                 ", LCD_LINE_2)
except:
  print("lcd screen is not correctly plugged in")

while(0 != GPIO.input(ButtonPin)):
    if (waitTime==0):
       startWaiting=time.time()
    waitTime=time.time()-startWaiting
    #print("waiting for user to push button... current wait time is",waitTime)
    if(waitTime>25):
          sadString="@@@@@"
          return sadString
    elif(waitTime>10):
          try:
             lcd_string("waiting for input...             ", LCD_LINE_1)
          except:
             print("lcd screen is not correctly plugged in")
    else:
          try:
             lcd_string("still waiting...                 ", LCD_LINE_1)
          except:
             print("lcd screen is not correctly plugged in")
waitTime=0


lcdButtonDisplay=""
lcd_string("Input detected:          ", LCD_LINE_1)
lcd_string(lcdButtonDisplay+"                            ", LCD_LINE_2)
time.sleep(lcdGlo[3])
lcd_string("Enter Morse Code:          ", LCD_LINE_1)


#lcdglo: lines=0, charPerLine=1, timeUnit=2, textDisplaySpeed=3, ledPin = 4, blinkOutputAi(on/off state)=5
```

```
    #SECTION 4: "SIMON SAYS" WHILE LOOP RECORDS BUTTON PUSHES UNTIL 14 TU PASS WITHOUT A PRESS=================================================================
    #============================================================================================

    dotDashMess='Message: '#these are the first 9 characters.  dot/dashes start on

    currentLineNum=0
    upperLcdLine=""

    while(OFFtimer<(14*lcdGlo[2])):#if user fails to blink light for 10 or more time units, our code
    #assumes he is done recording and ends the while loop, thereby proceeding on to the
    #next part where our code will plug the ON/OFF timer arrays into a PARROT array,
    #where the LED will parrot our own button pushes exactly.


    #============================================================================================
        if ((0 == GPIO.input(ButtonPin))):#When button is pushed this if loop runs
            GPIO.output(lcdGlo[4], GPIO.HIGH) #turns LED on

            if (OFFtimerStarted != 0):#all info gathered from the previously run elseloop goes here
                offList.append(OFFtimer)
                numButtonReleases=numButtonReleases+1
                if (OFFtimer>(3*lcdGlo[2])):
                    #print (dotDashMess + ' ')
                    #lcd_string(">>letter space<<                    ", LCD_LINE_1)
                    lcdButtonDisplay=lcdButtonDisplay+" "
                    currentLineNum=int((len(lcdButtonDisplay))/lcdGlo[1])
                    #lcd_string("Input detected:          ", LCD_LINE_1)
                    lcd_string(lcdButtonDisplay[(currentLineNum*lcdGlo[1]):((currentLineNum*lcdGlo[1])+lcdGlo[1])]+"",
LCD_LINE_2)

            OFFtimer=0
            OFFtimerStarted=0#the off timer values are all reset to 0


            if (ONtimerStarted==0):#recording the time at the moment of the button push
                startTimeOn=time.time()
                ONtimerStarted=1

            ONtimer= time.time() - startTimeOn
            #print("current time elapsed on button push ",numButtonPushes+1," is ",(ONtimer/lcdGlo[2]),"time units")
```

```
    #=============================================================================================
        else:#when button is NOT pushed
            GPIO.output(lcdGlo[4], GPIO.LOW) #turns LED off
            if (ONtimerStarted != 0):
                onList.append(ONtimer)#add our latest ONtimer recorded to our on array
                currentTimeOff=0
                upperLcdLine=""
                #I want to print out dash every time I get a dash and add that to the messageSoFar
                if (ONtimer>(3*lcdGlo[2])):
                    print (dotDashMess + '-')
                    lcdButtonDisplay=lcdButtonDisplay+"-"

                    currentLineNum=int((len(lcdButtonDisplay))/lcdGlo[1])

                    lcd_string(lcdButtonDisplay[(currentLineNum*lcdGlo[1]):((currentLineNum*lcdGlo[1])+lcdGlo[1])]+"",
LCD_LINE_2)

                else:
                    print (dotDashMess+ '.')
                    lcdButtonDisplay=lcdButtonDisplay+"."
                    currentLineNum=int((len(lcdButtonDisplay))/lcdGlo[1])
                    #lcd_string("Input detected:                ", LCD_LINE_1)
                    lcd_string(lcdButtonDisplay[(currentLineNum*lcdGlo[1]):((currentLineNum*lcdGlo[1])+lcdGlo[1])]+"",
LCD_LINE_2)
                numButtonPushes = numButtonPushes +1
                print("final time duration recorded for button push",numButtonPushes," is ",(ONtimer/lcdGlo[2])," time units")
            ONtimer=0
            ONtimerStarted=0#the on timer values are all reset to 0

            if (OFFtimerStarted==0):#recording the time at the moment of the button RELEASE
                startTimeOff=time.time()
                OFFtimerStarted=1

            if (OFFtimer<(1*lcdGlo[2]) and upperLcdLine != "-"):
                upperLcdLine="-"
                lcd_string("Time Units:"+str(upperLcdLine)+"                ", LCD_LINE_1)
            elif (OFFtimer>=(3*lcdGlo[2]) and upperLcdLine != "3" and upperLcdLine != ">=7"):
                upperLcdLine="3"
                lcd_string("Time Units:"+str(upperLcdLine)+"                ", LCD_LINE_1)
            elif (OFFtimer>=(7*lcdGlo[2]) and upperLcdLine != ">=7"):
                upperLcdLine=">=7"
                lcd_string("Time Units:"+str(upperLcdLine)+"                ", LCD_LINE_1)
```

```
        OFFtimer= time.time() - startTimeOff
        #print("current time elapsed SINCE button press number ",numButtonPushes+1," is ",OFFtimer/lcdGlo[2]," time units")

    #=============================================================================================
    #=============================================================================================
    #=============================================================================================
    #SECTION 5: FOR LOOP USED TO PARROT USER BUTTON INPUT (AND TRANSFER VALUES FROM ARRAYS INTO=====
    # STRINGS)-- it also performs 2 successive conversions with the aid of functions: first,========
    #
    #
    if (OFFtimerStarted != 0):
        offList.append(OFFtimer)#add one last recorded Off time (since we can't go back the other
    #loop to record it once its completed)
        numButtonReleases=numButtonReleases+1

    #We now have TWO arrays (lists): onList and offList

    count=0
    dd1=''#this is where our dots and dashes are stored-- it is illegal to have more than 5 in an english morse code char
    dd2=''
    dd3=''
    dd4=''
    dd5=''
    dd6=''
    ddCombo=''

    #print("debug check for index fixing: ")
    if((len(offList)) != (len(onList))):
      if((len(offList)) > (len(onList))):
        del offList[0]
      if((len(onList)) > (len(offList))):
        del onList[0]


    #writeToFile("",userSays1,"Morse Code"):

    for q in range(0, len(offList)):

        #print("onList value",q," is ", onList[q])
        #GPIO.output(lcdGlo[4], GPIO.HIGH) #turns LED on
        #time.sleep(onList[q])
```

```
    os.system('echo '+str(offList[q])+' >> bleepBloopTimes')
    os.system('echo '+str(onList[q])+' >> bleepBloopTimes')


    count=count+1


    if(onList[q]>=(3*lcdGlo[2])):#this means you've got a dash
        if (dd1 != '.' and dd1 != '-'):
            dd1='-'
        elif (dd2 != '.' and dd2 != '-'):
            dd2='-'
        elif (dd3 != '.' and dd3 != '-'):
            dd3='-'
        elif (dd4 != '.' and dd4 != '-'):
            dd4='-'
        elif (dd5 != '.' and dd5 != '-'):
            dd5='-'
        elif (dd6 != '.' and dd6 != '-'):
            dd6='-'
    else:
        if (dd1 != '.' and dd1 != '-'):
            dd1='.'
        elif (dd2 != '.' and dd2 != '-'):
            dd2='.'
        elif (dd3 != '.' and dd3 != '-'):
            dd3='.'
        elif (dd4 != '.' and dd4 != '-'):
            dd4='.'
        elif (dd5 != '.' and dd5 != '-'):
            dd5='.'
        elif (dd6 != '.' and dd6 != '-'):
            dd6='.'


    ddCombo=dd1+dd2+dd3+dd4+dd5+dd6
    print (ddCombo+' is our letter so far in dashes and dots')


    #print("offList value",q," is ", offList[q])
```

```
        #GPIO.output(lcdGlo[4], GPIO.LOW)#turns LED off
        #time.sleep(offList[q])

        if (count>6):
            someText = ddCombo+ ' is not a valid character in morse code.'
            dd1=''
            d=''
            dd3=''
            dd4=''
            dd5=''
            dd6=''
            ddCombo=''
            count=0
        if (offList[q]>=(3*lcdGlo[2])):
            convertedLetter=run_new_letter_funcs(ddCombo, convertedLetter)
            convertedWord=add_letter_to_word(convertedLetter,convertedWord)
            dd1=''
            dd2=''
            dd3=''
            dd4=''
            dd5=''
            dd6=''
            ddCombo=''
            count=0
            if (offList[q]>=(7*lcdGlo[2])):
                #lcd_string(">>WORD SPACE<<                    ", LCD_LINE_1)
                lcd_string(lcdButtonDisplay[(currentLineNum*lcdGlo[1]):((currentLineNum*lcdGlo[1])+lcdGlo[1])]+"",
LCD_LINE_2)
                convertedMessage=add_word_to_message(convertedWord,convertedMessage)
                convertedWord=''
                if (offList[q]>=(14*lcdGlo[2])):
                    #lcd_string(">>MESSAGE END.<<                    ", LCD_LINE_1)
                    convertedMessage=end_message(convertedMessage)
                    convertedWord=''
                    convertedMessage=convertedMessage[9:]
                    convertedMessage=convertedMessage[:-11]
                    convertedMessage2=convertedMessage
                    if(specFilter != ""):
                        convertedMessage= specifictionary(specFilter, convertedMessage)
                    if (convertedMessage[0]==" "):
                        convertedMessage2=convertedMessage[1:]
                    lcdSays("You entered: "+str(convertedMessage2))
                    return convertedMessage2
```

```
numCharLcdSet=lcdGlo[1]


def playModernJazz(genre,subgenre):
    #genre='Electronic'
    #subgenre='Chillout'
    subgenre=str(subgenre).lower()
    subgenre1=subgenre[0].upper()
    subgenre1=(subgenre1+subgenre[1:])
    subgenre=subgenre1
    #print('Our current subgenre is: ', subgenre)
    #playerReplySub=input("Check subgenre to see if correct")#human player selection process:


    #genre=str(genre).lower()
    genre1=genre[0].upper()
    genre1=(genre1+genre[1:])
    genre=genre1
    #print('Our current genre is: ', genre)
    #playerReplySub=input("Check genre to see if correct")#human player selection process:


    #omxplayer -o local /home/pi/Desktop/blueprintForFinalProjReconstruction/finalProjMusic/Jazz/modernJazz/electric-
tric_-_Sketch_26.mp3
    lcd_string("modern jazz              ", LCD_LINE_1)
    lcd_string("is available      ", LCD_LINE_2)

    stuffToDisplay="Please choose from the following list of songs: "
    #lcdSays(stuffToDisplay)

    fileContent=""
    listOfSongsFile="songsList"

    os.system('ls /home/pi/Desktop/finalProject/music/Electronic/Chillout/*.mp3 >
/home/pi/Desktop/finalProject/music/Electronic/Chillout/'+str(listOfSongsFile))
```

```
    #fileContent=readFromFile('/home/pi/Desktop/finalProject/music/Electronic/Chillout/'+listOfSongsFile)
    #lcdSays(fileContent)

    debugCount=2


    allSongsArray=['']
    allSongsArray=readFromFileArr('/home/pi/Desktop/finalProject/music/Electronic/Chillout/songsList')

    i=1
    numSongs=0
    numSongs=len(allSongsArray)
    fakeRandom=999
    fakeRandom=random.randint(1,(numSongs-1))


    randomSong=''
    cutSongArr=['']
    playerReply=""
    xxx="song list: "
    while(i<numSongs):
      cutSong1=''
      cutSong1=allSongsArray[i]

      startArr=0
startArr=cutSong1.find('/home/pi/Desktop/finalProject/music/Electronic/Chillout/')+len('/home/pi/Desktop/finalProject
/music/Electronic/Chillout/')
      cutSong2=''
      cutSong2=cutSong1[startArr:]
      cutSong3=''
      cutSong3=cutSong2[:-5]
      cutSongArr.append(cutSong3)
      #print(str(i)+') ')
      #lcdSays((str(i)+') '))
      #print(cutSong3)
      #if(randSongNum1==i):
        #randomSong=cutSong3
      #print(cutSongArr[i])


      ##################################################lcdSays(str(i)+'. '+str(cutSongArr[i]))
```

```
        xxx=(str(xxx)+"  "+str(i)+") "+str(cutSongArr[i]))
        #fakeRandom=7
        if(i==fakeRandom):
            randomSong=cutSong3
        i=i+1
    lcdSays(xxx)
    fakeRandom=7
    print('fakeRandom song is '+str(cutSongArr[fakeRandom]))

    #randomSong = cutSongArr[random.randint(1,(len(allSongsArray)-1))]

    menuSays="Choose from the song list above by entering the number of the song on the list: "
    morseReply="@@@@@"
    try:
        lcdSays(menuSays)
    except:
        print(menuSays)
    morseReply=""
    morseReply=morseCode(morseReply,"")
    playerReply=morseReply
    if(morseReply=="@@@@@"):
        playerReply=input("Choose from song list above using keyboard")#human player selection process:
        morseReply=playerReply
    try:
        songpath=cutSongArr[int(playerReply)]
    except:
        playerReply="1"
        songpath=cutSongArr[int(playerReply)]
    songNumAtStart=playerReply
    menuSays=("You chose: "+str(songpath))
    lcdSays(menuSays)
    #except:
    if (len(playerReply) > 3):
        songpath=cutSongArr[fakeRandom]
        menuSays=("No valid song choice selected... choosing a random song instead... Now Playing: "+str(songpath))
        lcdSays(menuSays)

    #stuffToDisplay=''
    #stuffToDisplay=('Random song chosen is '+str(randomSong))
```

```
    #lcdSays(randomSong)

    #songpath='Dee_Yan-Key_-_12_-_Birds'
    #songpath = str(randomSong)

    #songpath = ('/home/pi/Desktop/finalProject/music/Electronic/Chillout/'+randomSong)

    #songpath='/home/pi/Desktop/finalProject/music/Electronic/Chillout/Dee_Yan-Key_-_12_-_Birds.mp3'
    songpath='/home/pi/Desktop/finalProject/music/'+genre+'/'+subgenre+'/'+randomSong+'.mp3'

    omxprocess = subprocess.Popen(['omxplayer', '-o', 'local', songpath], stdin=subprocess.PIPE, stdout=None,
stderr=None, bufsize=0)

    time.sleep(5)

    print('NOspacing'+songpath+'NOspacing')
    print("debug count is",debugCount)

    time.sleep(.25)
    lcdSays("button push to end song coming up")

    fileContent=("/home/pi/Desktop/finalProject/music/"+genre+"/"+subgenre+"/currentKeyCommands")
    lcdSays(fileContent)

    morseReply=""
    morseReply=morseCode(morseReply,"")
    playerReply=morseReply

    nextSongNum2=0
    playerReplyU=playerReply.upper()
    defaultSongTimer=200
    curVol=0
    adjVol=0
    while(playerReplyU.count("Q")<=1):

        lcdSays("Choose from the following options to adjust music: D: Decrease volume   U: volume Up   Q: quits music   N:
next song   B: Back one song   P: Pause and Play   I: Increase speed   S: Slow down")
        if(inputMode == "Keyboard"):
            playerReply=input("Choose from the following options to adjust music: D: Decrease volume   U: volume Up   Q:
quits music   P: Pause and Play   I: Increase speed   S: Slow down")
        if((0 != GPIO.input(ButtonPin))):
            morseReply=morseCode(morseReply,"")
            playerReply=morseReply
        print("You entered: ",morseReply)
```

```
        playerReplyU=playerReply.upper()
        if (playerReplyU.count("Q")>=1):
            omxprocess.stdin.write(b'q')
            defaultSongTimer=0
            lcdSays("Quitting to main menu...")
            return
            time.sleep(1)
        if (playerReplyU.count("N")>=1):
            lcdSays("playing next song...")
            playerReplyU="Q"
            omxprocess.stdin.write(b'q')
            defaultSongTimer=0
            playerReplyU="N"
            if (playerReplyU.count("N")>=1):
                lcdSays("Playing Next Song...")
                nextSongNum2=nextSongNum2+1
                songpath=cutSongArr[int(songNumAtStart)+int(nextSongNum2)]
                lcdSays("Now Playing: "+str(cutSongArr[int(songNumAtStart)+int(nextSongNum2)]))
                fakeRandom=(int(songNumAtStart)+int(nextSongNum2))
                adjVol=-999
                i=1
                while(i<numSongs):
                    cutSong1=''
                    cutSong1=allSongsArray[i]

                    startArr=0

startArr=cutSong1.find('/home/pi/Desktop/finalProject/music/Electronic/Chillout/')+len('/home/pi/Desktop/finalProject
/music/Electronic/Chillout/')
                    cutSong2=''
                    cutSong2=cutSong1[startArr:]
                    cutSong3=''
                    cutSong3=cutSong2[:-5]
                    cutSongArr.append(cutSong3)
                    #print(str(i)+') ')
                    #lcdSays((str(i)+') '))
                    if(i==fakeRandom):
                        randomSong=cutSong3
                    i=i+1

                songpath='/home/pi/Desktop/finalProject/music/'+genre+'/'+subgenre+'/'+randomSong+'.mp3'
                omxprocess = subprocess.Popen(['omxplayer', '-o', 'local', songpath], stdin=subprocess.PIPE, stdout=None,
stderr=None, bufsize=0)
```

```
            time.sleep(3.5)
            adjVol==-999
            if(adjVol==-999):
               adjVol=0
            while (curVol>adjVol):
               omxprocess.stdin.write(b'+')
               time.sleep(1)
               adjVol=adjVol+1
            while (curVol<adjVol):
               omxprocess.stdin.write(b'+')
               time.sleep(1)
               adjVol=adjVol+1

      if (playerReplyU.count("B")>=1):
         lcdSays("playing previous song...")
         playerReplyU="Q"
         omxprocess.stdin.write(b'q')
         defaultSongTimer=0
         playerReplyU="N"
         if (playerReplyU.count("N")>=1):
            #lcdSays("Playing Next Song...")
            nextSongNum2=nextSongNum2-1
            songpath=cutSongArr[int(songNumAtStart)+int(nextSongNum2)]
            lcdSays("Now Playing: "+str(cutSongArr[int(songNumAtStart)+int(nextSongNum2)]))
            fakeRandom=(int(songNumAtStart)+int(nextSongNum2))
            i=1
            while(i<numSongs):
               cutSong1=''
               cutSong1=allSongsArray[i]

               startArr=0

startArr=cutSong1.find('/home/pi/Desktop/finalProject/music/Electronic/Chillout/')+len('/home/pi/Desktop/finalProject
/music/Electronic/Chillout/')
               cutSong2=''
               cutSong2=cutSong1[startArr:]
               cutSong3=''
               cutSong3=cutSong2[:-5]
               cutSongArr.append(cutSong3)
               #print(str(i)+') ')
               #lcdSays((str(i)+') '))
               if(i==fakeRandom):
                  randomSong=cutSong3
```

```
            i=i+1
        songpath='/home/pi/Desktop/finalProject/music/'+genre+'/'+subgenre+'/'+randomSong+'.mp3'
        omxprocess = subprocess.Popen(['omxplayer', '-o', 'local', songpath], stdin=subprocess.PIPE, stdout=None,
stderr=None, bufsize=0)
        time.sleep(5)
        adjVol==-999
        if(adjVol==-999):
            adjVol=0
        while (curVol>adjVol):
            omxprocess.stdin.write(b'+')
            time.sleep(1)
            adjVol=adjVol+1
        while (curVol<adjVol):
            omxprocess.stdin.write(b'+')
            time.sleep(1)
            adjVol=adjVol+1

    elif (playerReplyU.count("P")>=1):
        lcdSays("PAUSE PLAY")
        omxprocess.stdin.write(b'p')
    elif (playerReplyU.count("D")>=1):
        lcdSays("TURN VOLUME DOWN")
        omxprocess.stdin.write(b'-')
    elif (playerReplyU.count("U")>=1):
        lcdSays("TURN VOLUME UP")
        omxprocess.stdin.write(b'+')
    elif (playerReplyU.count("S")>=1):
        lcdSays("SLOW DOWN")
        omxprocess.stdin.write(b'1')
    elif (playerReplyU.count("I")>=1):
        lcdSays("INCREASE SPEED")
        omxprocess.stdin.write(b'2')
    playerReplyU=playerReply.upper()
```

```
inputMode=""


playerReply=""

morseReply=""


#MENU FROM END PLUGGED IN HERE %%%%%%% MENU FROM END PLUGGED IN HERE %%%%%%% MENU FROM END PLUGGED IN HERE %%%%%%% MENU FROM END PLUGGED IN HERE %%%%%%%%%%%%%%%%%
#||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
#vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv


#playModernJazz('ElecTroNic','CHILLOUT')

#welcomeMessage="Welcome to the Raspberry Pi alternate emergency user input system.  This interactive interface allows the computer and the user to communicate with one another in a variety of different manners to best suit the needs or constraints of a given situation.  The raspberry pi while this program is running is now capable of communicating via text through an LCD screen or through LED blinks and button pushes as Morse Code in addition to conventional user input methods."
lcd_init()

menuSays="Choose from options using Morse Code: "
lcdSays(menuSays)
mainMenuSays="Choose from the Following Menu Options : Option 1- Write to file   Option 2- read from file   Option 3- Perform online search   Option 4- check email   Option 5- send email   Option 6- execute terminal commands   Option 7- play music   Option 8- quit   Option 9- change settings"
lcdSays(mainMenuSays)
inputMode = "morseCode"

#lcdSays(welcomeMessage)
menuOptionPicked="99"

playerReplyU=""

while (playerReplyU.count("8")< 1):

  lcdSays(">>MAIN MENU<<")
  lcdSays("Enter a number 1-9 to select a menu option.  Enter 8 or q to quit.  Enter h for helpful hints.  Enter any other key to see menu options listed again.")
  morseReply=""
  playerReplyU=""
  if (inputMode != "Keyboard"):
```

```
    morseReply=morseCode(morseReply,"")
    playerReply=morseReply
  else:
    morseReply = "@@@@@"
    playerReply=input("Select one of the main menu options using the number keys on your keyboard:")
    lcdSays("Choose from MAIN MENU options using keyboard")
  if (morseReply != "@@@@@"):
    menuSays=("You entered: "+str(morseReply))

  #########  MENU OPTIONS ###################################

  playerReplyU=""
  playerReplyU=playerReply.upper()
  # create write to file
  if (playerReplyU.count("1")>=1):

      if (inputMode != "Keyboard"):
        lcdSays("First you need to enter a file name using Morse Code:")
        morseReply=morseCode(morseReply,"")
        fileNameIn=morseReply
      else:
        lcdSays("First you must enter a file name using keyboard:")
        time.sleep(lcdGlo[3])
        fileNameIn=input("Enter file name:")
      lcdSays("... now enter text with button or keyboard to edit text file: ")
      time.sleep(lcdGlo[3])

      writeToFile(fileNameIn,"",inputMode)
  #quiting from program
  elif (playerReplyU.count("8")>=1 or playerReplyU.count("Q")>=1 ):
      menuSays=("Quitting Morse Code program...")
      lcdSays(menuSays)
  elif (playerReplyU.count("H")>=1):
      menuSays=("Helpful Hints:")
      lcdSays(menuSays)
      lcdSays("1. Hold your button down for a second or so to skip over and past a long message")
      lcdSays("2. If you don't have a button available you can simply touch together the button pin which in your case is
currently"+str(ButtonPin)+" to any ground.  A simple paper clip or pair of wires or even just a metal chewing gum wrapper
will allow you to conduct electricity such that you can create your own button pin to skip through messages or enter
morse code")
      lcdSays("3. Since precise morse code input by a human is quite difficult our code is designed to be very easy-going with
the user and will extrapolate outwards to attempt to determine what the user entered even if it is not exact.  You need
not fret making small typos.  Just keep typing until you get something that our code might recognize and accept.  With
```

```
most of our code so long as the user enters a viable selection SOMEWHERE in his response our code will attempt to accomodate them.")


  elif (playerReplyU.count("2")>=1):
    # menuSays=("read from file is available at this time. ")
     #lcdSays(menuSays)
     if (inputMode != "Keyboard"):
       lcdSays("Enter file name using Morse Code:")
       morseReply=morseCode(morseReply,"")
       createdFileName=morseReply
     else:
       #morseReply=morseCode(morseReply,"")
       lcdSays("Enter file name using keyboard:")
       createdFileName=input("Enter file name:")
     try:
        fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
        lcdSays("The file called "+str(createdFileName)+" contains the following text: "+str(fileContent))
     except:
        fileContent=("This file could not be found-- Sorry!")
        #fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+"*"+str(createdFileName)+"*")
        lcdSays(fileContent)


     #online search
  elif (playerReplyU.count("3")>=1):
     menuSays=("enter a search term . ")
     lcdSays(menuSays)
     if (inputMode != "Keyboard"):
       morseReply=morseCode(morseReply,"")

       playerReply=morseReply
       fileContent=playerReply
     else:
       fileContent=input("Enter stuff to search online:")
     stuffFromWebsite=""
     fileContent=str(fileContent).lower()
     fileContent1=fileContent[0].upper()
     fileContent1=(fileContent1+fileContent[1:])
     #fileContent = fileContent.title()

     wikiString=fileContent1
     duckString=fileContent1
```

```
    webpageG=""
    lcdSays("First we will search for a wikipedia article on the search term you entered: ")
    try:
      webpageG = quickGrabAll(("https://en.wikipedia.org/wiki/"+str(wikiString)),webpageG)
      musicTrack=""
      musicTrack=(""+str(webpageG))
      endOfTrack=""
      endOfTrack="</p>"
      finalArray=""
      finalArray=finalTrackExtract(endOfTrack,musicTrack)
      lcdSays(finalArray)
    except:
      lcdSays("Sorry we could not find a wikipedia article on this subject")

    lcdSays(">>>Now we perform a secondary search using the duckduckgo search engine: ")
    webpageG = quickGrabAll(("https://duckduckgo.com/?q="+str(duckString)+"&t=h_&ia=about"),webpageG)
    musicTrack=""
    musicTrack=(""+str(webpageG))
    endOfTrack="Text\":\""
    finalArray=""
    finalArray=finalTrackExtract(endOfTrack,musicTrack)
    lcdSays(finalArray)


    #checking email
  elif (playerReplyU.count("4")>=1):
    menuSays=("checking email... ")
    lcdSays(menuSays)
    checkEmail()

    #sending emails
  elif (playerReplyU.count("5")>=1):
    menuSays=("Sending Email: ")
    lcdSays(menuSays)
    lcdSays("Enter email message text to send using morse code or keyboard: ")
    if (inputMode != "Keyboard"):
     morseReply=morseCode(morseReply,"")
     playerReply=morseReply
    else:
     playerReply=input("Enter email message: ")#human player selection process:
    menuSays=("You entered: "+str(playerReply))
```

```
    lcdSays(menuSays)

    email_sending('brentxavierproject@gmail.com',''+str(playerReply))
    #email_sending('brentxavierproject@gmail.com','blah bhlafdsafdsa')



    #terminal commands
  elif (playerReplyU.count("6")>=1):
    #menuSays=("terminal commands are available at this time. ")
    #lcdSays(menuSays)
    createdFile="stuff"
    createdFileName="listOfFiles"
    morseReply=""
    lcdSays("Enter safe terminal Commands from list of choices")
    lcdSays("1. ls: list all files in current directory  2. date: tell me the current time and date  3. mkdir: create a new directory within your current folder  4. cd: change our current default directory for this session to any subdirectory of our current directory")
    if (inputMode != "Keyboard"):
      lcdSays("Enter terminal command from list using button in morse code")
      morseReply=morseCode(morseReply,"")
      playerReply=morseReply
    else:
      playerReply=input("Enter terminal Command from list using keyboard: ")#human player selection process:
    print("You entered: ",playerReply)

    playerReplyU=playerReply.upper()
    if (playerReplyU.count("LS")>=1  or playerReplyU[0]=="1" or playerReplyU.count("1")>=1):
      os.system('ls -l /home/pi/Desktop/finalProject > /home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(createdFileName))
      #fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
      #print(fileContent)
      #lcdSays(fileContent)
      fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
      lcdSays(fileContent)
    elif (playerReplyU.count("CD")>=1 or playerReplyU[0]=="4"):
      createdFileName="defaultSub1"
      lcdSays("Choose a subdirectory: ")

      if (inputMode != "Keyboard"):
        morseReply=morseCode(morseReply,"")
        playerReply=morseReply
      else:
```

```
        playerReply=input("Enter subdirectory name: ")#human player selection process:
    menuSays=("You entered: "+str(playerReply))
    lcdSays(menuSays)
    playerReplyU=playerReply.upper()

    try:
      lcdGlo[6]=(str(playerReply)+"/")
      os.system('cd home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(createdFileName))
      createdFileName="timeAndDateCheck"
      os.system('date > /home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(createdFileName))
      fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
      lcdSays(fileContent)

    except:
      lcdSays("Sorry we are unable to change to the subdirectory you named")
      lcdGlo[6]=""
    #fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
    #print(fileContent)
    #lcdSays(fileContent)
    fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
    lcdSays(fileContent)

  elif (playerReplyU.count("DATE")>=1 or playerReplyU[0]=="2" or playerReplyU.count("2")>=1):
    createdFileName="timeAndDateCheck"
    os.system('date > /home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(createdFileName))
    #fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
    #print(fileContent)
    #lcdSays(fileContent)
    fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
    lcdSays(fileContent)
  elif (playerReplyU.count("DIR")>=1  or playerReplyU[0]=="3" or playerReplyU.count("MK")>=1):
    createdFileName="timeAndDateCheck"
    os.system('date > /home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(createdFileName))
    #fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
    #print(fileContent)
    #lcdSays(fileContent)
    fileContent=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
    fileC2=""
    fileC2=fileContent[:9]
    defaultName=""
    defaultName=("Folder"+str(fileC2))
    lcdSays("default created folder name is "+str(fileC2)+".  Enter Y or YES to use this name or N to create your own")
```

```
        if (inputMode != "Keyboard"):
          morseReply=morseCode(morseReply,"")
          playerReply=morseReply
        else:
          playerReply=input("Enter yes or no: ")#human player selection process:
        menuSays=("You entered: "+str(playerReply))
        lcdSays(menuSays)
        playerReplyU=playerReply.upper()

        if(playerReplyU[0]=="Y" or playerReplyU.count("YES")>=1):
          try:
            os.system('mkdir /home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(defaultName))
          except:
            lcdSays("sorry we were unable to create a new directory with this name")
        else:
          lcdSays("Ok then.  Enter a unique folder name now: ")
          if (inputMode != "Keyboard"):
            morseReply=morseCode(morseReply,"")
            playerReply=morseReply
          else:
            playerReply=input("Enter a unique name for our new directory: ")#human player selection process:
          menuSays=("You entered: "+str(playerReply))
          lcdSays(menuSays)
          try:
            os.system('mkdir /home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(playerReply))
          except:
            lcdSays("sorry we were unable to create a new directory with this name")


      else:
        lcdSays("Sorry that is not a valid selection.")


    # play music
    elif (playerReplyU.count("7")>=1):
      #menuSays=("music is available at this time. ")
      #lcdSays(menuSays)
      createdFile="stuff"
      createdFileName="subgenreList"
      #menuSays="Choose a subgenre from the following list of choices: "
      #lcdSays(menuSays)
```

```
    currentGenre='Electronic'
    currentSubgenre='Chillout'
    #os.system('ls /home/pi/Desktop/finalProject/music/'+str(currentGenre)+' > /home/pi/Desktop/finalProject/'+str(lcdGlo[6])+str(createdFileName))
    #menuSays=readFromFile("/home/pi/Desktop/finalProject/"+str(lcdGlo[6])+str(createdFileName))
    #lcdSays(menuSays)
    playModernJazz(currentGenre,currentSubgenre)
  elif (playerReplyU.count("9")>=1):
    lcdSays("Welcome to the Settings SubMenu: Please choose from the following list of choices to continue")
    settingsSubMenu=""
    lcdSays("Enter R to return to previous menu.  Otherwise enter a number to select one of the following options: ")
    settingsSubMenu="Option 1- Toggle Led Morse Code output mode  Option 2- Change Led GPIO pin  Option 3- change LCD screen characters per line  Option 4- change time unit length  Option 5- change text display speed  Option 6- return to main menu"
    lcdSays(settingsSubMenu)
    playerReplyU="1"
    morseReply=""
    if (inputMode != "Keyboard"):
      menuSays="Enter a submenu option using Morse Code: "
      lcdSays(menuSays)
      morseReply=morseCode(morseReply,"")
      playerReply=morseReply
    if (inputMode == "Keyboard"):
      morseReply = "@@@@@"
      lcdSays("Enter a submenu option number using keyboard")
      playerReply=input("Enter selection using keyboard here: ")
    if (morseReply != "@@@@@"):
      menuSays=("You entered: "+str(playerReply))
      playerReplyU=playerReply.upper()
    while (playerReplyU.count("1")>=1 or playerReplyU.count("2")>=1 or playerReplyU.count("3")>=1 or playerReplyU.count("4")>=1 or playerReplyU.count("5")>=1):
      playerReplyU=playerReply.upper()
      lcdSays(menuSays)
      if (playerReplyU.count("1")>=1):
        if(lcdGlo[5]==1):
          lcdGlo[5]=0
          lcdSays("led output turned OFF")
        elif(lcdGlo[5]==0):
          lcdGlo[5]=1
          lcdSays("led output turned ON")
      elif (playerReplyU.count("R")>=1):
        lcdSays("returning to main menu...")
        playerReply=""
      elif (playerReplyU.count("2")>=1):
```

```
            morseReply=""
            if (inputMode != "Keyboard"):
              menuSays="Enter a new LED GPIO pin location using Morse Code: "
              lcdSays(menuSays)
              morseReply=morseCode(morseReply,"")
              playerReply=morseReply
            if (inputMode == "Keyboard"):
              morseReply = "@@@@@"
              lcdSays("Enter a new LED GPIO pin location using keyboard")
              playerReply=input("Enter selection using keyboard here: ")
            if (morseReply != "@@@@@"):
              menuSays=("You entered: "+str(playerReply))
            playerReplyU=playerReply.upper()
            lcdSays(menuSays)
            if(playerReplyU.count("21")>=1):
                lcdGlo[4]=21
            elif(playerReplyU.count("12")>=1 and lcdGlo[7] != 12):
                lcdGlo[4]=12
            elif(playerReplyU.count("20")>=1):
                lcdGlo[4]=20
            elif(playerReplyU.count("26")>=1):
                lcdGlo[4]=26
            elif(playerReplyU.count("19")>=1):
                lcdGlo[4]=19
            elif(playerReplyU.count("25")>=1):
                lcdGlo[4]=25
          elif (playerReplyU.count("3")>=1):
            morseReply=""
            if (inputMode != "Keyboard"):
              menuSays="Enter the number of characters per line on your lcd screen using morse code: "
              lcdSays(menuSays)
              morseReply=morseCode(morseReply,"")
              playerReply=morseReply
            if (inputMode == "Keyboard"):
              morseReply = "@@@@@"
              lcdSays("Enter the number of characters per line on your lcd screen using keyboard: ")
              playerReply=input("Enter selection using keyboard here: ")
            if (morseReply != "@@@@@"):
              menuSays=("You entered: "+str(playerReply))
            playerReplyU=playerReply.upper()
            lcdSays(menuSays)
            if(playerReplyU.count("16")>=1):
```

```
            lcdGlo[1]=16
        elif(playerReplyU.count("20")>=1):
            lcdGlo[1]=20
        elif(playerReplyU.count("24")>=1):
            lcdGlo[1]=24
        elif(playerReplyU.count("25")>=1):
            lcdGlo[1]=25
        elif(playerReplyU.count("10")>=1):
            lcdGlo[1]=10
        elif(playerReplyU.count("11")>=1):
            lcdGlo[1]=11
        elif(playerReplyU.count("12")>=1):
            lcdGlo[1]=12
        elif(playerReplyU.count("14")>=1):
            lcdGlo[1]=14
        elif(playerReplyU.count("7")>=1):
            lcdGlo[1]=7
        elif(playerReplyU.count("8")>=1):
            lcdGlo[1]=8
        elif(playerReplyU.count("9")>=1):
            lcdGlo[1]=9
        else:
            lcdSays("No valid selection detected")
    elif (playerReplyU.count("4")>=1):
        morseReply=""
        lcdSays("Enter a number from 1 to 9 to select a new time unit length 0.1 to 0.9.  Your current timeUnit length is
"+str(lcdGlo[2]))
        if (inputMode != "Keyboard"):
          menuSays="Enter a new time unit length using Morse Code: "
          lcdSays(menuSays)
          morseReply=morseCode(morseReply,"")
          playerReply=morseReply
        if (inputMode == "Keyboard"):
          morseReply = "@@@@@"
          lcdSays("Enter a new time unit length using keyboard")
          playerReply=input("Enter selection using keyboard here: ")
        if (morseReply != "@@@@@"):
          menuSays=("You entered: "+str(playerReply))
        playerReplyU=playerReply.upper()
        lcdSays(menuSays)
        menuSays=""
        if(playerReplyU.count("1")>=1):
            lcdGlo[2]=0.1
```

```
            elif(playerReplyU.count("2")>=1):
                lcdGlo[2]=0.2
            elif(playerReplyU.count("3")>=1):
                lcdGlo[2]=0.3
            elif(playerReplyU.count("4")>=1):
                lcdGlo[2]=0.4
            elif(playerReplyU.count("5")>=1):
                lcdGlo[2]=0.5
            elif(playerReplyU.count("6")>=1):
                lcdGlo[2]=0.6
            elif(playerReplyU.count("7")>=1):
                lcdGlo[2]=0.7
            elif(playerReplyU.count("8")>=1):
                lcdGlo[2]=0.8
            elif(playerReplyU.count("9")>=1):
                lcdGlo[2]=0.9
            else:
                menuSays="No valid selection detected"
                lcdSays(menuSays)
            if(menuSays != "No valid selection detected"):
                lcdSays("Our new time unit length is "+str(lcdGlo[2]))
        elif (playerReplyU.count("5")>=1):
            morseReply=""
            lcdSays("Enter a number from 1 to 9 to select a new text display speed from 0.1 to 0.9.  The LARGER the number
the SLOWER the text will display.  Your current text display speed is "+str(lcdGlo[3]))
            if (inputMode != "Keyboard"):
              menuSays="Enter a new  text display speed using Morse Code: "
              lcdSays(menuSays)
              morseReply=morseCode(morseReply,"")
              playerReply=morseReply
            if (inputMode == "Keyboard"):
              morseReply = "@@@@@"
              lcdSays("Enter a new  text display speed using keyboard")
              playerReply=input("Enter selection using keyboard here: ")
            if (morseReply != "@@@@@"):
              menuSays=("You entered: "+str(playerReply))
            playerReplyU=playerReply.upper()
            lcdSays(menuSays)
            menuSays=""
            if(playerReplyU.count("1")>=1):
                lcdGlo[3]=0.1
            elif(playerReplyU.count("2")>=1):
                lcdGlo[3]=0.2
```

```
            elif(playerReplyU.count("3")>=1):
                lcdGlo[3]=0.3
            elif(playerReplyU.count("4")>=1):
                lcdGlo[3]=0.4
            elif(playerReplyU.count("5")>=1):
                lcdGlo[3]=0.5
            elif(playerReplyU.count("6")>=1):
                lcdGlo[3]=0.6
            elif(playerReplyU.count("7")>=1):
                lcdGlo[3]=0.7
            elif(playerReplyU.count("8")>=1):
                lcdGlo[3]=0.8
            elif(playerReplyU.count("9")>=1):
                lcdGlo[3]=0.9
        else:
            lcdSays("No valid input detected.  Please try again.")

        lcdSays("Enter R to return to the main menu")
        if (inputMode != "Keyboard"):
            menuSays="Enter a submenu option using Morse Code: "
            lcdSays(menuSays)
            morseReply=morseCode(morseReply,"")
            playerReply=morseReply
            playerReplyU=playerReply.upper()
        if (inputMode == "Keyboard"):
            morseReply = "@@@@@"
            lcdSays("Enter a submenu option number using keyboard")
            playerReply=input("Enter selection using keyboard here: ")
            playerReplyU=playerReply.upper()
        if (morseReply != "@@@@@"):
            menuSays=("You entered: "+str(playerReply))
            playerReplyU=playerReply.upper()
    lcdSays("returning to main menu...")



### here is where our main menu attempts to adapt to accomodate the user's potential lack of a button by giving the option of switching to keyboard:
  else:
    if(inputMode != "Keyboard"):
      menuSays=("No valid Morse Code input detected.  Your current default button pin used to enter Morse Code is GPIO"+str(ButtonPin)+".")
```

```
    lcdSays(menuSays)
    lcdSays("Enter K to switch to keyboard mode enter a number with keyboard to change GPIO pin location.  Enter S to skip.")
    playerReply=input("Enter K to switch to keyboard mode or enter a number with keyboard to change GPIO pin location.  Enter S to skip.")
    playerReplyU=playerReply.upper()
    if((playerReplyU.count("K")>=1)):
      inputMode="Keyboard"
      lcdSays("Switching to keyboard input mode now...")
    elif (playerReplyU.count("12")>=1):
      ButtonPin=12
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    elif (playerReplyU.count("14")>=1):
      ButtonPin=14
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    elif (playerReplyU.count("15")>=1):
      ButtonPin=15
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    elif (playerReplyU.count("16")>=1):
      ButtonPin=16
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    elif (playerReplyU.count("18")>=1):
      ButtonPin=18
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    elif (playerReplyU.count("27")>=1):
      ButtonPin=27
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    elif (playerReplyU.count("22")>=1):
      ButtonPin=22
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    elif (playerReplyU.count("10")>=1):
      ButtonPin=10
      lcdGlo[7]=ButtonPin
      lcdSays(("Changing button pin to GPIO"+str(lcdGlo[7])))
    else:
      lcdSays("No valid choice detected.")
```

```
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(ButtonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)



    ButtonPin=lcdGlo[7]
    lcdSays(menuSays)
    mainMenuSays="Choose from the Following Menu Options : Option 1- Write to file   Option 2- read from file   Option 3- Perform online search   Option 4- check email   Option 5- send email   Option 6- execute terminal commands   Option 7- play music   Option 8- quit   Option 9- change settings"
    lcdSays(mainMenuSays)

  #quiting from program
  if (playerReplyU.count("8")>=1):
    menuSays=("Quitting Morse Code program...")
    lcdSays(menuSays)




#^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
#||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
#MENU FROM END PLUGGED IN HERE %%%%%%% MENU FROM END PLUGGED IN HERE %%%%%%% MENU FROM END PLUGGED IN HERE %%%%%%% MENU FROM END PLUGGED IN HERE %%%%%%%%%%%%%%%%%%
```