IFB102, Introduction to Computer Systems
Module 3: Mini-Project

“MusicBox”

Student Name: Samuel Fleming

Student Number: n10755306

Tutor(s):
Neco Kriel [n.kriel@qut.edu.au],
Ignatius Chuckwudi [ignatius.chukwudi@qut.edu.au]

Class Allocation: Thursday, 8:00am

Statement of Contribution

By completing this form **I agree to the following**:

1. I declare that all of the work submitted for this assignment is my own original work except for material that is explicitly referenced and for which I have permission, or which is freely available (and also referenced).
2. I agree that QUT may archive this assignment for an indefinite period of time, and use it in the future for educational purposes including, but not limited to: as an example of previous work; as the basis for assignments, lectures or tutorials; for comparison when scanning for plagiarism, etc.
3. I agree to indemnify QUT and hold it blameless if copyright infringements are found in this work and the copyright owner takes action against QUT that is not covered by the normal terms of Educational Use.

Project Objectives

Problem Definition:
Creating pitched music can be a very tedious experience for beginners. This is due to theory such as; note placement, scales and chord selection needing to be understood when creating melodies and chord progressions. There is a demand for specific services/devices to enable theoretically correct music creation for those uneducated with the theory.

The goal of this project “MusicBox” is; *to create a physical interface (device) to allow un-educated musicians to create theoretically correct music*. Based on the investigation into associated technologies (discussion), and method presented in the project; “gpio music box” (https://projects.raspberrypi.org/en/projects/gpio-music-box), a device will be created to fulfill the stated goal. The “gpio music box” project created by Dr Sally Le Page, published on raspberrypi.org is the inspiration for this project. “MusicBox” created by Samuel Fleming will be an extension to that of Le Page’s creation with iterative development process of ‘Replicate and Extend’.

The functional objectives of the final version of “MusicBox” device were defined as follows:

- Produce an audio projections of pitched notes are upon button press
- Pitched notes (and respective button) are ordered in a musically acceptable key
- Upon button press, the note will be visually displayed to the user
- Users can select from an inventory of scales
- Users can select a specified base-note of the selected scale

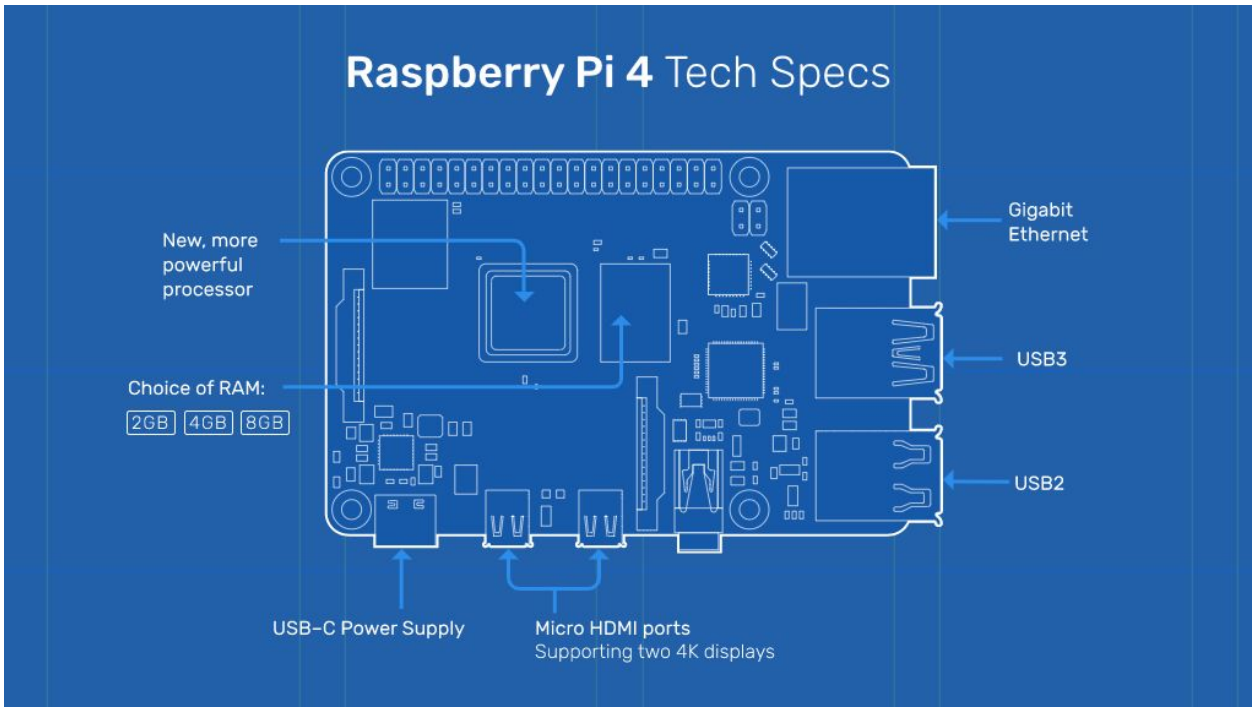
Technological Discussion

Pre-implementation investigation occurred addressing the following information systems topics and their degree of implementation into the project: Hardware (Raspberry Pi, Visual Display, GPIO, Breadboard - content), Operating Systems (Raspbian), Networking (Bluetooth) and Languages, Libraries and Complexity (Pygame, Python).

Hardware

Hardware has been defined as; “collection of physical parts of a computer system.” Some common examples of computer hardware are listed as follows: monitor, keyboard, mouse, motherboard, hard drive, disk drive etc *(Study.com, 2020).*

The model of raspberry pi used in the project was a Raspberry Pi 4. The inbuilt hardware components of this model can be shown in the following diagram.



Investigation of Specific Hardware Components And Their Projected Implementation:

HDMI (Visual Display):

HDMI is an audio/visual data transferring interface which can be implemented on the raspberry Pi. It transfers uncompressed video and audio (uncompressed or compressed) digital data from a source device (transmitter) to a compatible visual monitor or audio device (or device that outputs both.) In the project, HDMI will be implemented to transfer the raspberry Pi’s GUI into a

visual display to appear on a monitor. The raspberry Pi has two micro HDMI output terminals and with a standard HDMI cable and adapter, can be connected to a monitor. An alternative to HDMI for achieving visual display on a raspberry Pi includes usage of a GPIO to USB cable where using a windows computer and software such as putty, the linux command line of the raspberry Pi can be accessed. The main concern for this alternate method is that it does not allow for the linux GUI to be used. *(Wikipedia.org, 2020).*

General Purpose Input/Output (gpio), Breadboard and Buttons:

GPIO pins are an array of small metal pins on the raspberry pi which can conduct electricity. Each gpio pin has its own identifier and the Raspberry pi can fire commands when it detects a flow of current through a specific gpio pin. Buttons act as switches in an electric circuit. When un-pressed, a button is an open switch allowing no current flow. When pressed, the button allows immediate current flow. Electric inputs (such as; buttons) can connect to gpio pins and send current and electric messages to the raspberry pi which can activate commands. In the same sense, the raspberry pi can send commands to specific gpio pins which can activate output devices (such as speakers and LED's). Gpio pins are a "male" connector and therefore require female infrastructure to transmit electrical signals. *(Le Page, 2019)*.

Breadboards are an array of rows and columns of female electric connecting points being places where male connectors can be inserted. The breadboard conducts electricity down a negative and positive column, and depending on the size of the breadboard, conducts electricity through numerous rows of female points. In these rows is where buttons are placed to correspond to GPIO pins.

The only alternative to a system of breadboards, wiring and buttons corresponding to GPIO pins to execute commands would be to implement a graphical user interface of digital buttons and inputs. An example would be the implementation of the GUi API "anvil" in the raspberry pi *(upperlevel/anvil-gui, 2020)*. But in the case of a music box, GUI button inputs could potentially undermine the user experience of a physical user interface of music playing that buttons would produce.

Operating Systems

(OS) can be defined as; "software that communicates with the hardware and allows other programs to run. It consists of system software, or the fundamental files your computer needs to boot up and function". *(Techterms.com, 2020)* OS's comprises compilers and default files/programs to allow the machine code (and hardware components) to interact with the user in a friendlier way. A term used to describe this is; 'complexity gap'. *(lecture 3)*

Raspbian OS (Raspberry Pi Operating System):

Raspbian OS is the standard Operating system used in Raspberry Pi 4 and is an adaptation of the linux operating system. A major characteristic of the linux operating system is that it provides

users with access to the source code (command line to the CPU kernel). Access to the kernel allows control of allocation of resources, processes and memory which provides many advantages including; detection of bugs and tracking of resources. *(EDUCBA. 2020)*. In the implementation of the project, Raspbian OS also provides the default program “Thonny” which is inbuilt with software libraries including functions; “pygame” and “gpiozero” to allow effective processing with the hardware interface (breadboard, buttons and GPIO pins). Thonny and its libraries are a mandatory component in replicating and extending the predessessing project (Dr Sally Le Page) and therefore no alternatives for Raspbian were considered. *(Le Page, 2019)*.

Networking

Networking is the term used to describe the methodology of transmitting data between nodes (data processing objects) and is the basis for many developments such as the IoT. No implications of networking were mentioned in the predessessing project (Dr Sally Le Page). *(lecture 4)*.

Bluetooth (Keyboard and Mouse):

Bluetooth is described as being not-network-oriented, but is commonly implemented in short range wireless connections of devices such as mice, keyboards and audio devices. Bluettoh operates over radio frequencies approximate to 2.45GHz. *(Wikipedia.org, 2020)*

When Raspbian OS is operating a GUI, mouse and keyboards are very helpful in navigating the screen. Keyboard and mice pairings embedded with bluetooth can be implemented into the setup of the raspberry Pi system. Bluetooth keyboard and mouse pairings come with a usb drive embedded with the bluetooth receiving technology to receive transmissions made by both the mouse and keyboard. This is effective as the only one of the four Pi 4 USB ports will be capacitated. Alternatively a mouse and a keyboard are both individually connected with wires. This method mitigates latency experience with the Bluetooth transmission but leaves less free USB ports.

Languages, Libraries and Complexity

Python (Programming Language):

“Python is a programming language that lets you work quickly and integrate systems more effectively” *(python.org).* Python is an advanced general purpose third generational (3GL) programming language which can be found as the default scripting language in the Raspberry Pi 4’s default programming software “Thonny”. Python is considered to be an advanced 3GL as it has the capabilities of existing in software libraries with 4GL which most 3GL’s cannot do. A Collection of specified python components (functions and concepts) have been listed and described as follows.

- **Variables;** used to store values. Variable values can be obtained, updated or removed and if the variable is numerical it can experience mathematical operatives and tests.

- **Lists (arrays);** are used to store a group of values rather than a single variable. Lists do this by storing the values in a specified order each with a specified numerical index in the list. When referencing a singular value in a list the function must identify the specific index and therefore specific indexes of lists can be written to and values updated.
- **For loops;** For loops are functions where specified instructions are repeated a set amount of times. This amount of times could be specified as a variable (integer) value. Loops in python are very applicable to operating with lists as the function could repeat as many times as there are indexes in a specified list and therefore in each loop of the function, a different list index (and its corresponding value) be stored and operated with. For loops can be embedded in functions or have clauses (such as if statements or other food loops) embedded in it.
- **If/else statements;** Possibly the most commonly used clause in python. The if/else statement tests a relationship between a variable and a stated value, or between two variables, and executes instructions depending on whether the relationship is true or not. If/else statements can be embedded within statements (just like for loops) and can also be embedded in itself in a continuing else/if format.
  (Wikipedia.org)

Pygame (Software Library):

Pygame is a set of modules of functions closely related to the SDL (Simple DirectMedia Library) software library. Intended for game development, Pygame functions are very applicable to gaming interfaces such as buttons and their implementation of multi-core CPU's allows for numerous processes to run at once. Pygame is very applicable to being implemented in recreational raspberry Pi programs with functions such as "gpiozero" and "Button[(gpio no.)]", and pygame is imported into the script in Le Page's "gpio music box" program.

Design and Implementation

First Iterative Prototype (Replication)

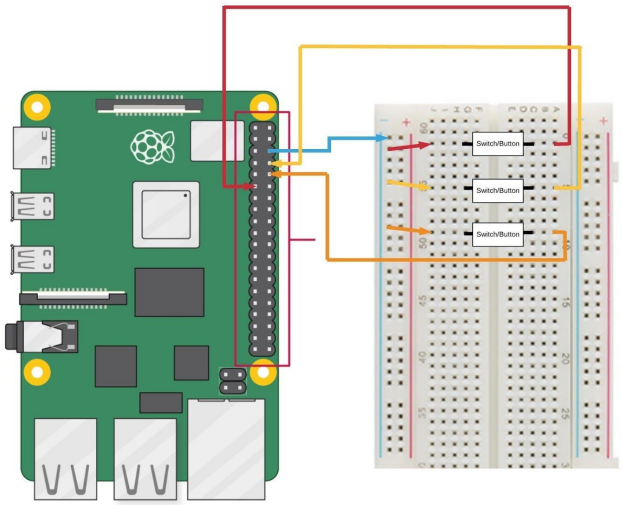
The first iterative prototype is a replica of the existing project; “GPIO Music Box” published found at: https://projects.raspberrypi.org/en/projects/gpio-music-box. This was implemented with the use of the raspberry Pi’s general purpose input/output pins (gpio), breadboard wiring, buttons/switches, Raspbian OS and Python Programming language.

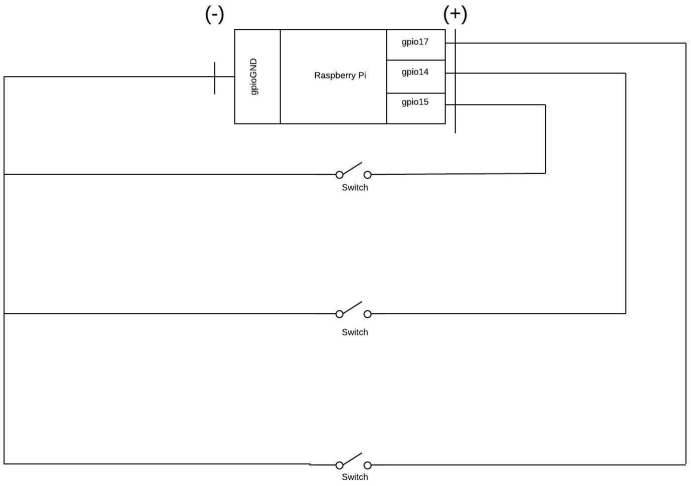
Prototype Objective(s):

- (In general) replicate the project; “gpio music box” created by Dr Sally Le Page. Therefore:
    - establish a reliable user Interface of buttons, and corresponding (controlled) sounds.

Hardware Orientation:

- (refer to implementation video addressing; HDMI, audio and mouse/keyboard)





All Code (`Python`):

```
from gpiozero import Button

import pygame

pygame.init()
```

```
sndRed =
pygame.mixer.Sound('/home/pi/Desktop/Music_Samples/piano-g_G_major.wa
v')
sndYellow =
pygame.mixer.Sound('/home/pi/Desktop/Music_Samples/piano-f_F_major.wa
v')
sndOrange =
pygame.mixer.Sound('/home/pi/Desktop/Music_Samples/piano-eb_D#_major.
wav')

btnRed = Button(17)
btnYellow = Button(14)
btnOrange = Button(15)

def red():
    print("red has been pressed (G)")
    sndRed.play()

def yellow():
    print("yellow has been pressed (F)")
    sndYellow.play()

def orange():
    print("orange has been pressed (D#)")
    sndOrange.play()

btnRed.when_pressed = red
btnYellow.when_pressed = yellow
btnOrange.when_pressed = orange
```

Description:

Using a male-to-female wire, the negative terminal on the breadboard was connected to the “ground/GND” gpio pin on the raspberry pi to power the breadboard. From the negative stream, 3 parallel male-to-male wires in the breadboard each consisting of a switch/button. From here the circuit(s) were connected to respective gpio terminals 17, 14 and 15. Using python, using functions from “pygame” SL, the raspberry pi would play a specified “.wav” audio file and produce a “print” comment upon detecting current from the respective gpio terminal (when its switch was closed/button pressed). This created a UI of 3 buttons producing 3 separate sounds upon button press.

Evaluation:

The prototype allowed users to play notes G, F and D# allowing for melodies such as “hot cross buns to be played”. The prototype successfully achieved its stated objective of; “establish a reliable user Interface of buttons, and corresponding (controlled) sounds.” Some extension ideas/suggestions for the prototype included; more buttons/notes to be pressed, easier UI of pressing buttons (wires were often in the way).

Second Iterative Prototype (Extend - part 1)

Prototype Goal(s):

- Establish pitched ordering of 8 notes in button correspondence
- Allow user to change between scales
- Allow for easier pressing of buttons

Description:

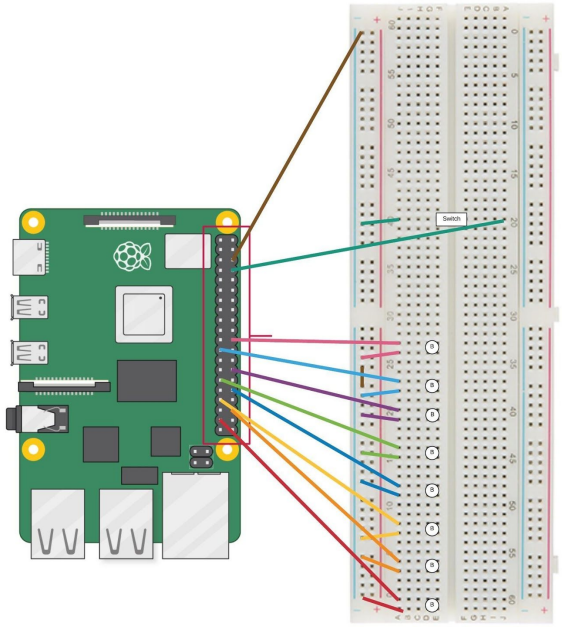
The breadboard was rearranged with the intention for better usability as it was declared in prototype 1 that switches were hard to push. This time instead of 3 switches, 8 buttons were implemented into the gpio pins. An additional switch/button as added onto the breadboard to act as a toggle to move between scale options in the scale inventory. Using integer variables and if/else statements, upon pressing the scale button, the integer representation from the scale was changed between the values of 0(major), 1(minor) and 2 (Phrygian Dominant). The functions corresponding to each button press contained if/else statements that tested the value of the scale integer variable.

Variable Inventory:

- Scales: MAjor (scale = 0), Minor (scale = 1), Phrygian Dominant (scale = 2)

Hardware Orientation:

- (refer to implementation video addressing; HDMI, audio and mouse/keyboard)



Code Snippets (repetition in code was excluded from report):

```
from gpiozero import Button
import pygame

pygame.init()

sndA = pygame.mixer.Sound('/home/pi/Desktop/Music.Samples/Piano/piano_A.wav')
sndAs = pygame.mixer.Sound('/home/pi/Desktop/Music.Samples/Piano/piano_A#.wav')
sndB = pygame.mixer.Sound('/home/pi/Desktop/Music.Samples/Piano/piano_B.wav')
```

[this occurred for 5 more ‘snd’ variables]

```
scale,key = 0,1

btn_one = Button(26)
btn_two = Button(16)
btn_three = Button(13)
btn_four = Button(12)
btn_five = Button(5)
btn_six = Button(1)
btn_seven = Button(11)
btn_eight = Button(25)
btn_key = Button(17)
btn_scale = Button(14)

def scale_change():
    global scale
    if scale < 2:
        scale+=1
        print(scale)
    else:
        scale = 0
        print(scale)
btn_scale.when_pressed = scale_change

def one():
    sndA.play()
    print("A")
btn_one.when_pressed = one

def two():
    global scale
    if scale == 0:
        sndB.play()
```

```
        print("B")
    elif scale == 1:
        sndB.play()
        print("B")
    else:
        sndAs.play()
        print("A#")
btn_two.when_pressed = two
```

[repetitive use of if statements continued with 5 more functions]

Evaluation:
The prototype successfully allowed users to play notes ordered into a selected scale by the user in the key of A. There were more notes and they were easier to play which was a great improvement from prototype 1. This allowed an exploration of melodies within the key and chords to be played. Some negative observations and ideas for extension for “MusicBox” included; poor communication of the scale selection (tell the user which scale is currently selected), allow selection of key base notes (keys), minimise the amount of repetition in code.

Third Iterative Prototype (Final Version of “MusicBox”) (Extend - part 2)

Prototype Goal(s):

- Establish an inventory of scales and key-notes
- Establish an algorithm enabling input for scale selection and key-note selection to select a list of notes from the inventory of key notes to act as the 8 playable buttons.
- Establish a user guide and UI comments

Developer Goals:

- Avoid repetition, write a more efficient script and algorithm

Description:
Using lists, scale note displacement for each of the three scales (major minor and Phrygian Dominant) were stored in the default key of A. Also lists were used as storage for the string value (“A”, “A#”, “B” …… & “Major”, “Minor” etc) equivalents for integer variables; scale_num, key_num (where the integer variable acted as the index). Using file names as string variables (dynamic list) and a dynamic list for the button’s corresponding notes, a for-loop-centralised algorithm was constructed. The algorithm operated such that the list tracking the notes corresponding to each of the buttons was updated every time either the scale_num was changed (by the user) or the key_num was changed (by the user). Incrementing of these variable was performed with if/else statements (as was in prototype two and variable ‘scale’). In more specific detail the algorithm is justified with the use of developer comments (#) in the python script.

Inventory:

- Keys: A, A#, B, C, C#, D, D#, E, F, F#, G, G# (12)

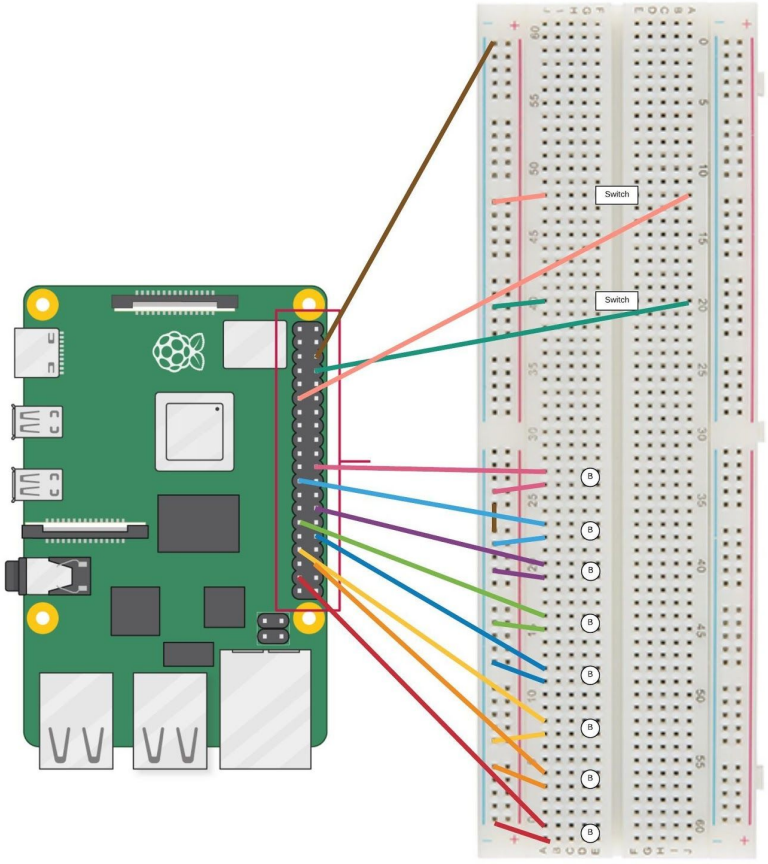
0	1	2	3	4	5	6	7	8	9	10	11
A	A#	B	C	C#	D	D#	E	F	F#	G	G#

- Scales: Major, Minor, Phrygian Dominant (3)

0	1	2
Major	Minor	Phrygian Dominant

<u>Hardware Orientation:</u>

- (refer to implementation video addressing; HDMI, audio and mouse/keyboard)



All Code (Python):

- Developer comments (#) implemented to justify algorithm logic

```
from gpiozero import Button

import pygame

pygame.init()

scale_num, key_num = 0, 0

#Arrays of integer values corresponding values:
key_nom = ["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#"]
scale_nom = ["Major", "Minor", "Phyrigian Dominant"]

#dynamic array:
array_dynam = [0, 2, 4, 5, 7, 9, 11]

#array for sound file name storage
snd = ["0-" + str(array_dynam[0]) + ".wav", "0-" + str(array_dynam[1]) + ".wav"
, "0-" + str(array_dynam[2]) + ".wav" , "0-" + str(array_dynam[3]) + ".wav",
"0-" + str(array_dynam[4]) + ".wav" , "0-" + str(array_dynam[5]) + ".wav", "0-"
+ str(array_dynam[6]) + ".wav"]

#default scale arrays:
def_0 = [0, 2, 4, 5, 7, 9, 11]
def_1 = [0, 2, 3, 5, 7, 8, 10]
def_2 = [0, 1, 4, 5, 7, 8, 10]

btn_one = Button(26)
btn_two = Button(16)
btn_three = Button(13)
btn_four = Button(12)
btn_five = Button(5)
btn_six = Button(1)
btn_seven = Button(11)
btn_eight = Button(25)
```

```
btn_key = Button(17)
btn_scale = Button(14)
btn_instrument = Button(15)

#Definition of default sound file variables
snd1 = pygame.mixer.Sound('/home/pi/Desktop/MusicBoxProject/sounds/' + snd[0])
snd2 = pygame.mixer.Sound('/home/pi/Desktop/MusicBoxProject/sounds/' + snd[1])
snd3 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" + snd[2])
snd4 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" + snd[3])
snd5 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" + snd[4])
snd6 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" + snd[5])
snd7 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" + snd[6])

#introductory comments
print("")
print("--------------------------------------------------------------------------
---------------")
print("")
print("Welcome to the scale generator tool made by Samuel Fleming; an interface
for exploring the different sounding musical pitches and scales.")
print("")
print("To play sounds, press any combination of the eight coloured buttons.")
print ("")
print("The current key setting is; A major, but feel free to press either
buttons to the right of the coloured buttons to toggle through our inventory of
key-pitches and scale types. ")



#main configuration function
def scale_config():

      #setting as global variables for processing
      global scale_num
      global key_num
      global array_dynam
      global current_scale
      global def_0
```

```
	global def_1
	global def_2
	global snd1
	global snd2
	global snd3
	global snd4
	global snd5
	global snd6
	global snd7

	#if statement to determine current scale list/array
	if scale_num == 0:
	current_scale = def_0
	elif scale_num == 1:
	current_scale = def_1
	else:
	current_scale = def_2

	 # for loop calculates an enters a new value in array_dynam for as many times as the length of the lis (seven times)
	for x in range(0, len(current_scale)):

	current_value = current_scale[x]
	#adding the key_num to default scale values to calculate dynamic array values.
	if (current_value + key_num) < 12:
		array_dynam[x] = (current_value + key_num)
	else:
		array_dynam[x] = (current_value + key_num - 12)
	#variable put into the sound file arrays
	snd[x] = "0-" + str(array_dynam[x]) + ".wav"



	print("")
	print("The Buttons are now set to the scale/key of; " + key_nom[key_num] + " " + scale_nom[scale_num])
```

```
	#updating each sound variable via index value in the list/array called
'snd'
	snd1 = pygame.mixer.Sound('/home/pi/Desktop/MusicBoxProject/sounds/' +
snd[0])
	#print("Button one plays; " + '/home/pi/Desktop/MusicBoxProject/sounds/'
+ snd[0])
	snd2 = pygame.mixer.Sound('/home/pi/Desktop/MusicBoxProject/sounds/' +
snd[1])
	#print("Button one plays; " + '/home/pi/Desktop/MusicBoxProject/sounds/'
+ snd[1])
	snd3 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" +
snd[2])
	#print("Button one plays; " + '/home/pi/Desktop/MusicBoxProject/sounds/'
+ snd[2])
	snd4 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" +
snd[3])
	#print("Button one plays; " + '/home/pi/Desktop/MusicBoxProject/sounds/'
+ snd[3])
	snd5 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" +
snd[4])
	#print("Button one plays; " + '/home/pi/Desktop/MusicBoxProject/sounds/'
+ snd[4])
	snd6 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" +
snd[5])
	#print("Button one plays; " + '/home/pi/Desktop/MusicBoxProject/sounds/'
+ snd[5])
	snd7 = pygame.mixer.Sound("/home/pi/Desktop/MusicBoxProject/sounds/" +
snd[6])
	#print("Button one plays; " + '/home/pi/Desktop/MusicBoxProject/sounds/'
+ snd[6])


#function for change in scale
def scale_change():
	global scale_num
	if scale_num < 2:
	scale_num+=1
	else:
	scale_num = 0
	current_scale_nat = scale_nom[scale_num]
	print ("")
```

```
print("--------------------------------------------------------------------------
---------------")
      print("*Excecuted A Scale Change (Scale Change Button Pressed)*")
      print("")
      print ("The nature of the key (Scale type) has changed to; " +
scale_nom[scale_num] + ". Press to change again.")


      #run the main configuration function to update dynamic aray everytime a
variable is changed by user
      scale_config()
#scale changes whenever 'btn_scale' is pressed
btn_scale.when_pressed = scale_change


#function for change in scale
def key_change():
      global key_num
      if key_num < 11:
      key_num+=1
      else:
      key_num = 0
      current_key = key_nom[key_num]
      print ("")

print("--------------------------------------------------------------------------
---------------")
      print("*Excecuted A Key Change (Key Change Button Pressed)*")
      print("")
      print ("The pitch of the key (Key Bass Note) has changed to; " +
key_nom[key_num] + ". Press to change again.")


      #run the main configuration function to update dynamic array every time a
variable is changed by user
      scale_config()
#scale changes whenever 'btn_scale' is pressed
btn_key.when_pressed = key_change


#function(s) corresponding to each button press:
```

```
def one():
      snd1.play()
      print("")
      print(key_nom[key_num] + " " + scale_nom[scale_num] + "; " +
key_nom[array_dynam[0]])
btn_one.when_pressed = one


def two():
      snd2.play()
      print("")
      print(key_nom[key_num]+ " "  + scale_nom[scale_num] + "; " +
key_nom[array_dynam[1]])
btn_two.when_pressed = two


def three():
      snd3.play()
      print("")
      print(key_nom[key_num]+ " "  + scale_nom[scale_num] + "; " +
key_nom[array_dynam[2]])
btn_three.when_pressed = three


def four():
      snd4.play()
      print("")
      print(key_nom[key_num]+ " "  + scale_nom[scale_num] + "; " +
key_nom[array_dynam[3]])
btn_four.when_pressed = four


def five():
      snd5.play()
      print("")
      print(key_nom[key_num]+ " "  + scale_nom[scale_num] + "; " +
key_nom[array_dynam[4]])
btn_five.when_pressed = five


def six():
      snd6.play()
```

```
      print("")
      print(key_nom[key_num]+ " "  + scale_nom[scale_num] + "; " +
key_nom[array_dynam[5]])
btn_six.when_pressed = six


def seven():
      snd7.play()
      print("")
      print(key_nom[key_num]+ " "  + scale_nom[scale_num] + "; " +
key_nom[array_dynam[6]])
btn_seven.when_pressed = seven


btn_eight.when pressed = one
```

Evaluation (Presented as Strengths & Limitations):

Strengths	Limitations
- Effectively projects audio corresponding to a correct ordering of piano notes upon button press - Effective use of lists (dynamic and static) and incremental variables - Accurately changes the scale-nature upon button press - Accurately changes key-base-note upon button press - Effectively guides user with abundance of comments and 'prints' - The coding for the program being variable-based allows it to be very adaptable (For example; an additional scale could easily be added into the script) - Very limited knowledge/skill in music is required to operate MusicBox at a high level.	- Inventory of scales was small at only 3 - User has to interact with the source code which can be aesthetically unpleasant - User may have to deal with wiring, GPIO stability and buttons which can be inconsistent and off-putting - Device had an instrument inventory of only a piano - Slight delay between button press and audio output

Project Extension Possibilities (Ways that the final prototype of "MusicBox" could be extended - Suggestions):

- Add an inventory of .wav files of numerous instruments and implement a feature allowing for the user to select the instrument sound they want to use
- Instead of the use of a breadboard and button, use key binds on a keyboard. This would allow for a larger amount of playable buttons and notes and a better user experience exploring scales and keys.
- Increase the inventory of scales and keys, this would greater improve the user experience and effectiveness of the project outcome
- Implement a GUI for more effective user experience in selecting scales, keys and potentially instruments

References (Technological Investigation):

EDUCBA. 2020. *Linux Vs Windows - Find Out The 9 Most Awesome Differences*. [online] Available at: <https://www.educba.com/linux-vs-windows/> [Accessed 27 March 2020].

En.wikipedia.org. 2020. *Bluetooth*. [online] Available at: <https://en.wikipedia.org/wiki/Bluetooth> [Accessed 27 March 2020].

Raspberry Pi. 2020. *Download Raspberry Pi OS For Raspberry Pi*. [online] Available at: <https://www.raspberrypi.org/downloads/raspbian/> [Accessed 29 May 2020].

En.wikipedia.org. 2020. *Python (Programming Language)*. [online] Available at: <https://en.wikipedia.org/wiki/Python_(programming_language)> [Accessed 29 May 2020].\

Techterms.com. 2020. *Operating System Definition*. [online] Available at: <https://techterms.com/definition/operating_system> [Accessed 29 May 2020].

GitHub. 2020. *Upperlevel/Anvil-Gui*. [online] Available at: <https://github.com/upperlevel/anvil-gui> [Accessed 29 May 2020].

En.wikipedia.org. 2020. *HDMI*. [online] Available at: <https://en.wikipedia.org/wiki/HDMI> [Accessed 29 May 2020].

Study.com. 2020. [online] Available at: <https://study.com/academy/lesson/what-is-computer-hardware-components-definition-examples.html> [Accessed 29 May 2020].