Sentience

2.1

Generated by Doxygen 1.8.14

Thu Apr 19 2018 20:51:03

Contents

1 Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

5 Namespace Documentation

5.1 Sentience Namespace Reference

Classes

- class ActionInput
- class DeleteDialog
- class PrintDialog
- class SentienceApp
- class SentienceScreen
- class sentienceScreenManager

Variables

- string __author__ = 'Aaron Johnson'
- string __copyright__ = 'Copyright (c) 2018 Copyright Holder All Rights Reserved.'
- string __license__ = 'MIT'
- string __version__ = '2.1'
- string __maintainer__ = 'Aaron Johnson'
- string __email__ = 'Aaronjohnson@protonmail.ch'
- root_widget = Builder.load_string()

5.1.1 Variable Documentation

5.1.1.1 __author__

```
string Sentience.__author__ = 'Aaron Johnson'  [private]
```

Definition at line 33 of file Sentience.py.

5.1.1.2 __copyright__

```
string Sentience.__copyright__ = 'Copyright (c) 2018 Copyright Holder All Rights Reserved.'
[private]
```

Definition at line 34 of file Sentience.py.

5.1.1.3 __email__

```
string Sentience.__email__ = 'Aaronjohnson@protonmail.ch'  [private]
```

Definition at line 38 of file Sentience.py.

5.1.1.4 __license__

```
string Sentience.__license__ = 'MIT'  [private]
```

Definition at line 35 of file Sentience.py.

5.1.1.5 __maintainer__

```
string Sentience.__maintainer__ = 'Aaron Johnson'  [private]
```

Definition at line 37 of file Sentience.py.

5.1.1.6 __version__

```
string Sentience.__version__ = '2.1'  [private]
```

Definition at line 36 of file Sentience.py.

5.1.1.7 root_widget

```
Sentience.root_widget = Builder.load_string()
```

Definition at line 6194 of file Sentience.py.

5.2 SettingsMenu Namespace Reference

Variables

- my_settings

5.2.1 Variable Documentation

5.2.1.1 my_settings

```
SettingsMenu.my_settings
```

Definition at line 3 of file SettingsMenu.py.

6 Class Documentation

6.1 Sentience.ActionInput Class Reference

Inheritance diagram for Sentience.ActionInput:

6.2 Sentience.DeleteDialog Class Reference

Inheritance diagram for Sentience.DeleteDialog:

Collaboration diagram for Sentience.DeleteDialog:

Static Public Attributes

- delete_file = ObjectProperty(None)
- Cancel = ObjectProperty(None)

6.2.1 Detailed Description

```
DeleteDialog(FloatLayout):

Parameters
----------
    param1 : FloatLayout
        This is pretty much exactly what it looks like. When this
        is used later on it will automatically add a float layout.
Attributes
----------
    delete_file
        This will be used along side an ObjectProperty to register
        it for use with the SentienceScreen.delete_file() and
        SentienceScreen().open_delete_file_dialog() functions.

    Cancel
        This will be used along side an ObjectProperty to register
        it for use with the SentienceScreen.dismiss_popup() function.

    ObjectProperty(None)
        Initializes the two attributes to ObjectProperty. This is
        a built in feature of kivy to reduce code and make it
        easier to create/manipulate/initialize/instantiate
        both variables and functions. By making these two
        attributes object properties, in this case, we're
        literally binding them to the two functions calls
        listed above.
Members
-------
    None
Private Members
---------------
    None
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
    We use this with our popup window for deleting specific files.
    This is our dialog. A FloatLayout is provided by default and
    two other layouts are added to it in the kv design language.

    The ObjectProperty delete_file refers to a SentienceScreen()
    function: SentienceScreen.open_delete_file_dialog(). Clicking
    the button "Delete File" calls the open_delete_file_dialog()
    function which then opens a popup window.

    The ObjectProperty Cancel refers to the the button "Cancel"
    which is contained in the above mentioned popup window.
```

Definition at line 86 of file Sentience.py.

6.2.2 Member Data Documentation

6.2.2.1 Cancel

```
Sentience.DeleteDialog.Cancel = ObjectProperty(None)  [static]
```

Definition at line 141 of file Sentience.py.

6.2.2.2 delete_file

```
Sentience.DeleteDialog.delete_file = ObjectProperty(None)  [static]
```

Definition at line 140 of file Sentience.py.

The documentation for this class was generated from the following file:

- Sentience.py

6.3 Sentience.PrintDialog Class Reference

Inheritance diagram for Sentience.PrintDialog:

Collaboration diagram for Sentience.PrintDialog:

Static Public Attributes

- print_files = ObjectProperty(None)
- Cancel = ObjectProperty(None)

6.3.1 Detailed Description

```
PrintDialog(FloatLayout):

Parameters
----------
param1 : FloatLayout
    The first parameter. Will hold the widgets in the Popup window which
    creates a PrinterDialog window. Allowing the user to navigate to and
    select a file for printing.

Attributes
----------
print_files = ObjectProperty(None)
    print_files binds to the SentienceScreen().print_files() function.

Cancel = ObjectProperty(None)
    Cancel binds to the SentienceScreen().dissmis_popup() function.

Members
-------
    None
Private Members
---------------
    None
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
This class is essentially a container for the Popup() that's created
in SentienceScreen() class. The purpose of the Popup() is to allow
the user to have a graphical window to navigate to, and select from,
a list of files that they want to print out. Rather than automatically
printing out the files for the user. This prevents potential issues
and also allows the user the freedom to print out different files
created by this program.
```

Definition at line 41 of file Sentience.py.

6.3.2 Member Data Documentation

6.3.2.1 Cancel

```
Sentience.PrintDialog.Cancel = ObjectProperty(None)  [static]
```

Definition at line 83 of file Sentience.py.

6.3.2.2 print_files

```
Sentience.PrintDialog.print_files = ObjectProperty(None)  [static]
```

Definition at line 82 of file Sentience.py.

The documentation for this class was generated from the following file:

- Sentience.py

6.4 Sentience.SentienceApp Class Reference

Inheritance diagram for Sentience.SentienceApp:

Collaboration diagram for Sentience.SentienceApp:

Public Member Functions

- def build (self)
- def load_settings (self)
- def build_config (self, config)
- def build_settings (self, settings)
- def on_start (self)
- def on_stop (self)
- def warning_removal (self, dt)
- def set_username (self, value)
- def set_gender (self, value)
- def set_age (self, value)
- def on_config_change (self, config, section, key, value)

Public Attributes

- title
- settings_cls
- sentience
- use_kivy_settings
- is_audio_on
- increased_rate_of_speech
- decreased_rate_of_speech
- is_voice_on
- print_status
- delete_file_status
- delete_all_status
- user_exists
- gender_exists
- age_exists
- clear_screen_status
- file_creation_status
- profiler

6.4.1 Detailed Description

Definition at line 6319 of file Sentience.py.

6.4.2 Member Function Documentation

6.4.2.1 build()

```
def Sentience.SentienceApp.build (
              self )
```

```
    Kivy App() class is the core class that creates the
     main window and runs the program.

     You'll also see that I've created a series of variables
     which are used to store the data contained in
     Sentience.ini

     The data stored in these variables is loaded into them
     everytime that this program is run so that they always
     reflect the accurate settings.
```

Definition at line 6321 of file Sentience.py.

6.4.2.2 build_config()

```
def Sentience.SentienceApp.build_config (
              self,
              config )
```

We call this to build the menu. We're establishing default values which we'll then load in from our json string. This is what creats the actuall widgets and links them to the keys, provides default values, and registers them.

Definition at line 6436 of file Sentience.py.

6.4.2.3 build_settings()

```
def Sentience.SentienceApp.build_settings (
              self,
              settings )
```

Here we add the settings panel as widget in the form of a json object. We pass it the name of our panel, Sentience Settings, our self.config from build_config and the json strings which contains all of the data in self.config. This setups, create and adds the settings panel to the screen.

Definition at line 6462 of file Sentience.py.

6.4.2.4 load_settings()

```
def Sentience.SentienceApp.load_settings (
              self )
```

This is fairly straightforward. When the program launches we look at the data contained in Sentience.ini which holds the values from our Ssettings panel. Everytime that the user changes a setting it's written to Sentience.ini

Based on those settings we set or ignore specific variables from SentienceScreen(). We use this to maintain continuity between the end of the program and it being restarted.

Definition at line 6357 of file Sentience.py.

Here is the call graph for this function:

6.4.2.5 on_config_change()

```
def Sentience.SentienceApp.on_config_change (
              self,
              config,
              section,
              key,
              value )
```

```
    Simple vent monitor. Every time that the user changes
    a setting on the Settings panel, this function is called.

    It recevies the section (menu), the key (specific option),
    and the value which is what the current option has been
    changed to. From there we change the variables in the
    SentienceScreen() class.
```

Definition at line 6551 of file Sentience.py.

Here is the call graph for this function:

6.4.2.6 on_start()

```
def Sentience.SentienceApp.on_start (
              self )
```

```
    This function creates a cProfiler() to help us
    diagnose potential issues.
```

Definition at line 6475 of file Sentience.py.

6.4.2.7 on_stop()

```
def Sentience.SentienceApp.on_stop (
              self )
```

```
    When the program is exited the profiler is stopped
    and the .profile file containing the data that's
    been accumulated to help test the program is output
    to a file named SentiencePRofile.profile
```

Definition at line 6485 of file Sentience.py.

6.4.2.8 set_age()

```
def Sentience.SentienceApp.set_age (
              self,
              value )
```

```
    We call this function to set the age supplied to
    value when the users modifies the age setting in
    the settings menu. We store the gender in value in
    self.sentience.user_priofile[2]. We then clear the
    self.sentience.master_log string to ensure a new
    experience has been created for the current user.
```

Definition at line 6537 of file Sentience.py.

Here is the caller graph for this function:

6.4.2.9 set_gender()

```
def Sentience.SentienceApp.set_gender (
              self,
              value )
```

```
    We call this function to set the gender supplied to
    value when the users modifies the Gender setting in
    the settings menu. We store the gender in value in
    self.sentience.user_priofile[3]. We then clear the
    self.sentience.master_log string to ensure a new
    experience has been created for the current user.
```

Definition at line 6523 of file Sentience.py.

Here is the caller graph for this function:

6.4.2.10 set_username()

```
def Sentience.SentienceApp.set_username (
              self,
              value )
```

```
    This function is called to set the first key of
    self.sentience.user_profile[1] dictionary to
    value. value is then stored in self.sentience.username
    self.sentience.master_log string is then cleared to
    ensure a new user experience is created. We then create
    the user profile which basically just reads the users
    input username which is stored in value.
```

Definition at line 6506 of file Sentience.py.

Here is the caller graph for this function:

6.4.2.11 warning_removal()

```
def Sentience.SentienceApp.warning_removal (
              self,
              dt )

    A simple function to clear the contents of
    self.sentience.ids.view_port Widget.
```

Definition at line 6497 of file Sentience.py.

6.4.3 Member Data Documentation

6.4.3.1 age_exists

```
Sentience.SentienceApp.age_exists
```

Definition at line 6349 of file Sentience.py.

6.4.3.2 clear_screen_status

```
Sentience.SentienceApp.clear_screen_status
```

Definition at line 6350 of file Sentience.py.

6.4.3.3 decreased_rate_of_speech

```
Sentience.SentienceApp.decreased_rate_of_speech
```

Definition at line 6342 of file Sentience.py.

6.4.3.4 delete_all_status

```
Sentience.SentienceApp.delete_all_status
```

Definition at line 6346 of file Sentience.py.

6.4.3.5 delete_file_status

```
Sentience.SentienceApp.delete_file_status
```

Definition at line 6345 of file Sentience.py.

6.4.3.6 file_creation_status

`Sentience.SentienceApp.file_creation_status`

Definition at line 6351 of file Sentience.py.

6.4.3.7 gender_exists

`Sentience.SentienceApp.gender_exists`

Definition at line 6348 of file Sentience.py.

6.4.3.8 increased_rate_of_speech

`Sentience.SentienceApp.increased_rate_of_speech`

Definition at line 6341 of file Sentience.py.

6.4.3.9 is_audio_on

`Sentience.SentienceApp.is_audio_on`

Definition at line 6340 of file Sentience.py.

6.4.3.10 is_voice_on

`Sentience.SentienceApp.is_voice_on`

Definition at line 6343 of file Sentience.py.

6.4.3.11 print_status

`Sentience.SentienceApp.print_status`

Definition at line 6344 of file Sentience.py.

6.4.3.12 profiler

`Sentience.SentienceApp.profiler`

Definition at line 6480 of file Sentience.py.

6.4.3.13 sentience

`Sentience.SentienceApp.sentience`

Definition at line 6337 of file Sentience.py.

6.4.3.14 settings_cls

`Sentience.SentienceApp.settings_cls`

Definition at line 6335 of file Sentience.py.

6.4.3.15 title

`Sentience.SentienceApp.title`

Definition at line 6334 of file Sentience.py.

6.4.3.16 use_kivy_settings

`Sentience.SentienceApp.use_kivy_settings`

Definition at line 6338 of file Sentience.py.

6.4.3.17 user_exists

`Sentience.SentienceApp.user_exists`

Definition at line 6347 of file Sentience.py.

The documentation for this class was generated from the following file:

- Sentience.py

6.5 Sentience.SentienceScreen Class Reference

Inheritance diagram for Sentience.SentienceScreen:

Collaboration diagram for Sentience.SentienceScreen:

Public Member Functions

- def __init__ (self, kwargs)
- def display_user_conversation (self)
- def increase_chatbot_volume (self, vol)
- def decrease_chatbot_volume (self, vol)
- def set_volume (self, vol)
- def get_user_text_response (self)
- def increase_rate_of_speech (self, value)
- def decrease_rate_of_speech (self, value)
- def get_caprica_text_response (self)
- def get_user_voice_response (self)
- def get_caprica_voice_response (self, words)
- def set_gender (self)
- def set_speech_rate (self)
- def caprica_speak (self, words)
- def onEnd (self, name, completed)
- def clear_viewport (self)
- def create_user_profile (self)
- def set_enable_disable_audio (self)
- def set_enable_disable_voice (self)
- def print_files (self, path, filename)
- def create_dir (self, path)
- def write_logs (self)
- def open_print_file_dialog (self)
- def dismiss_popup (self)
- def on_mouse_pos (self, instance, pos)
- def display_tooltip (self, args)
- def close_tooltip (self, dt)
- def set_tooltip_text (self, text)
- def caprica_timer (self, _time)
- def start_timer_thread (self, _time)
- def check_timer (self, _time)
- def get_caprica_response (self)
- def get_caprica_voice_thread (self, words)
- def start_get_response_thread (self)
- def start_voice_response_thread (self)
- def get_user_text (self)
- def open_delete_file_dialog (self)
- def delete_file (self, path, filename)
- def delete_all (self)

Public Attributes

- tooltip_open
- tooltip
- engine
- record
- mic
- chatbot
- audio_threshold
- master_log
- voice_enabled
- voice_disabled

- user_input
- audio_enabled
- audio_disabled
- user_profile
- username
- current_conversation

Private Member Functions

- def __create_files (self, path)
- def __append_file (self, words, path)
- def __set_thinking_text (self, bool)
- def __currently_thinking (self, bool)

Private Attributes

- __is_thinking
- _popup

6.5.1 Detailed Description

```
SentienceScreen(Screen):

Parameters
----------
    param1 : Screen
        The first parameter creates a new Screen, which will function
        as a "page". This page is our only "Screen". It's the Main
        Window. It does everything. Now the actual designer code is
        done in the kv design language. But, this widget holds it
        all. It's the core of the program.

Attributes
----------
    self.chatbot
        The chatbot is the core feature here. It's the bot that the
        user communicates with. It's initialized and trained in the
        __init__ function. It's training can be continued throughout
        the program. Or expanded on by creating and adding new databases
        to its training regiment.

    self.engine
        The engine object refers to the python3 text to speeh engine
        . It's what enables the chat bot to have a voice. From this
        engine we derive the ability to pass a string to the chat
        bot which can then access the systems text to speech software
        and read it back with an apropriate voice.

    self.record
        The record object comes from speech_engine.Recognizer().
        This object allows us the ability to use programs such as
        CMU Sphinx voice recognition. Essentially we use this to
        transcribe recored audio to text which we can then store
        in a string. I make use of this by transcribing the recored
        audio to string vairables and passing them to the chat bot
        so that it can accurately respond to the user.

    self.mic
        This object allows us to access and use any connected or
        onboard microphone if one is available. With this we can
        record a users voice, store it in a variable then send it
        to the Recognizer() to be transcribed and passed as a string
```

```
        to the chat bot.

    self.audio_threshold
        This is used to automatically set the level at which the
        microphone accepts audio input. The higher the level the
        less sensitive the microphone is. Or rather the it's less
        likely that ambient noise will be treated as intentional
        audio being sent through the microphone.

    self.record_dynamic_energy_threshold
        This applies to self.record and is a boolean variable. By
        setting this to False we can ensure that the energy_threshold
        doesn't dynamically set its energy_threshold level. Note:
        That the energy_threshold is what enables us to searate
        between ambient noise and the users intended voice commands.

    self.master_log
        This is a string variable that I use to store all of the
        conversation that takes place between the user and the chat
        bot.

    self.voice_enabled
        If self.voice_enabled is set to True then the user is able
        to use their microphone to communicate with the chat bot.
        Note: The user can only use a microphone if they have one.
        This can be either a connected microphone and or an onboard
        microphone.

    self.voice_disabled
        If self.voice_disabled is set to True then the user can only
        communicate with the chat bot through text. Note: The chat
        bot can access its audio functions even if
        self.voice_disabled == True. This function only effects the
        users ability to use their microphone.

    self.user_input
        This is a string variable which I use to store the input
        from the user the data here is passed to the chat bot,
        stored in various files and variables/data structures.
        Note: This variable is redundant and will in the future be
        removed. It can be ommited and replaced by the TextInput
        widgets return function.

    self.audio_enabled
        if self.audio_enabled == True the chat bot can use the systems
        text to speech software (espeak, spai5, or nsss) to access the
        softwares built in voices and read back any strings that the
        chat bot comes up with as a response to the user. Note: This
        boolean vairable only effects the chat bots ability to use
        sound as a medium for communication. It does not effect the
        users ability to use their microphone.

    self.audio_disabled
        If self.audio_disabled == True then the chat bot can only
        communicate with the user via text.

    self.__user_profile
        self.__user_profile is a dictionary and stores three specific
        keys. 1) Username, 2) Age, 3) Gender. These are optional
        variables. The user doesn't need to create a user profile.
        Though it's encouraged that they do for better logging of
        the data. Note: If the user elects to not create a user profile
        this information is by default set.


Members
-------
    def __init__(self, **kwargs)
        Initalizes SentienceScreen() a more in depth analysis will
        be given under the SentienceScreen().__init__(self, **kwargs)
        functions documentation.

    def quick_check_os(self)
```

```
        This function is called when the user clicks on the
        "Check Operating System" button which is represented by
        an image of a computer on the menu bar. This function
        when clicked checks to see if the user is running either
        windows or Linux. If the user is running windows it makes
        three new TextInput Widgets visible by changing the opacity.
        If the user is using a Linux operating system clicking on
        this button does nothing. A more in depth analysis will be
        given in the SentienceScreen().quick_check_os() functions
        documentation.

    def get_user_text_response(self)
        This function is called when the user hits the "enter key"
        on their keyboard while inside of the user_input TextInput
        Widget. A string variable is returned from this and passed
        to the chat bot so that it can form a response to what the
        users statement was. A more in depth analysis of this will
        be given in the SentienceScreen().get_user_text_response()
        functions documentation.

    def get_caprica_text_response(self)
        This function is called after the user inputs a text
        response. And that response is sent to the chat bot. The
        response that the user input is used by this function to
        generate a response from the chat bot. A more in depth
        analysis will be given in the
        SentienceScreen().get_caprica_text_response() functions
        documentation.

    def get_user_voice_response(self)
        This function is called when the user clicks the
        "Record user" button. Which is located on the menu bar and
        is represented by the image of a blue talking head. If
        self.voice_disabled == True then the image will be a red
        talking head. If the user clicks the button when it's red a
        warning message will be displayed informing the user that
        he/she needs to first enable their microphone by clicking on
        the set_enable_disable_voice button. A More in depth
        analysis of this function will be given in the
        SentienceScreen().get_caprica_voice_response() function
        documentation.

    def get_caprica_voice_response(self, words)
        This function is called after the user inputs a text string
        in the proper TextInput widget; or
        if self.voice_enabled == True. A more in depth analysis of
        this function will be given in the
        SentienceScreen().get_caprica_voice_response(self, words)
        function documentation.

    def set_gender(self):
        This function is called in
        SentienceScreen().__init__(self, **kwargs). Through this
        function we set the voice property of self.engine to use
        the systems female voice option. A more in depth analysis
        of this function will be given in the
        SentienceScreen().set_gender(self) function documentation.

    def set_speech_rate(self):
        This function is called in
        SentienceScreen().__init__(self, **kwargs). Through this
        function we can set the self.engine speech rate property.
        This function can in effect lower or increase the number
        of words spoken by the chat bot per minute. A more in
        depth analysis of this function will be given in the
        SentienceScreen().set_speech_rate() functions
        documentation.

    def caprica_speak(self, words)
        This function is called from a variety of locations for the
        purpose of activating the voice feature of the chat bot
        which is derived from self.engine. A more in depth analysis
        of this function will be given in the
```

```
    SentienceScreen().caprica_speak(self, words) functions
    documentation.

def onEnd(self, name, completed)
    This function is called everytime
    self.caprica_speak(self, words) is called. This function is
    fired when the self.caprica_speak event has ended. This is a
    callabck which terminates the event queue of the
    self.engine. A more in depth analysis of this function will
    be given in the
    SentienceScreen().onEnd(self, name, completed) functions
    documentation.

def clear_viewport(self)
    This function is caleld whenever the user clicks the
    "Erase logs" button. Which is represented by the eraser on
    the menu bar. This button only erases the text in the
    viewport TextInput Widget. A more in depth analysis of
    this function will be given in
    SentienceScreen().clear_viewport(self) function
    documentation.

def create_user_profile(self)
    This function is highly redundant and will be removed in
    the future. This function is called when ever the user inputs
    their username for the first time. It runs some checks and
    then simply calls self.caprica_speak() to speak the users
    input username. A more in depth analysis of this function
    will be given in the
    SentienceScreen().create_user_profile(self) function
    documentation.

def set_enable_disable_audio(self)
    This function is called when the user clicks the
    self.set_enable_disable_audio button which is represented by
    either a red or blue speaker image on the menu bar. If
    self.audio_enabled == True the chat bot can use audio to
    communicate with the user and the image is a blue speaker.
    If self.audio_disabled == True then the chat bot can only
    communicate with the user via text. The button is  also
    then represented by a red speaker. This function will
    update the image on the menu bar to reflect its current
    status. A more in depth analysis of this function will
    be given in the SentienceScreen().set_enable_disable_audio(self)
    function documentation.

def set_enable_disable_voice(self)
    This function is called when the user clicks the
    self.set_enable_disable_voice button which is represented by
    either a red or blue microphone image on the menu bar.
    If self.voice_enabled == True the user can use their microphone
    to communicate with the chat bot and the image is a blue
    microphone. If self.voice_disabled == True then the user can
    only communicate with the chat bot via text. The button is
    also then represented by a red microphone. This function
    will update the image on the menu bar to reflect its current
    status. A more in depth analysis of this function will be
    given in the SentienceScreen().set_enable_disable_voice(self)
    function documentation.

def set_username(self)
    This function is called from two locations both involve the
    user inputting a desired username into a TextInput Widget
    and hitting the "Enter" key on their keyboard. This function
    sets the user name for the current user and can be changed
    at any time. A more in depth analysis of this function will
    be given in SentienceScreen().set_username() function
    documentation.

def set_sex(self)
    This function is called from two locations both involve the
    user inputting their gender into a TextInput Widget and
    hitting the "Enter" key on their keyboard. This function sets
```

```
    the gender for the current user and can be changed at any time.
    A more in depth analysis of this function will be given in
    SentienceScreen().set_sex() function documentation.

def set_age(self)
    This function is called from two locations both involve the
    user inputting their age into a TextInput Widget and hitting
    the "Enter" key on their keyboard. This function sets the users
    age for the current user and can be changed at any time. A more
    in depth analysis of this function will be given in
    SentienceScreen().set_username() function documentation.

def print_files(self, path, filename)
    This function is called when the user clicks on the "Print"
    button on the menu bar. When called a Popup() window is
    created and allows the user to navigate to any file that
    they wish to print. within that window are two buttons.
    Clicking the "Print" button will print the selected file
    while clicking the "Close" button will close the Popup()
    window. A more in depth analysis of this function will be
    given in SentienceScreen().print_files(self, path, filename)
    function documentation.

def create_dir(self, path)
    This function is caleld from within
    SentienceScreen().__init__(self, **kwargs). When executed it
    checks to see if a specific system relative directory exists.
    If it does the function returns nothing. If it doesn't exist
    the function creates the directory and then calls the private
    function self.__create_files(self, path). A more in depth
    analysis of this function will given in
    SentienceScreen().create_dir(self, path) function
    documentation.

def write_logs(self)
    This function is caleld when the user clicks the "Write Logs"
    button on the menu bar which is represented by a pencil
    image. It creates and writes the contents of self.master_log
    to a text file which is either
    "Users input username + _Conversations"
    .txt or simply "Username_Conversations".txt.
    A more in depth analysis of this function will be given in
    SentienceScreen().write_logs(self) function documentation.

def open_print_file_dialog(self)
    This function is caleld when the user clicks the "Print"
    button on the menu bar. This is the function that calls
    the Popup() window and allows the user to print a specific
    chosen file after navigating to it; and then by clicking the
    "Print files" button on that Popup() window.

def dismiss_popup(self)
    This function is called when the user clicks the "close"
    button on the PrintDialog() Popup() window. It closes the
    Popup() window. A more in depth analysis of this function
    will be given in the SentienceScreen().dismiss_popup()
    function documentation.

def on_mouse_pos(self, instance pos):
        This function is called everytime that the user moves
        his or her mouse. If the mouse collides with any of the
        the buttons on the menu bar (Action Bar) this function
        checks the positions against the various if statements
        which relate to the specific button. When the position
        of the users mouse matches the positions outlined in
        the statements. A tool tip is displayed, which presents
        at leat the name of the button.

def display_tooltip(self, *args):
    When this function is called the tooltip that relates
    to the button (as explain in on_mouse_pos) is created
    and added to the users screen. A clock event is then
    scheduled to delete the tooltip from the screen automaticaly
```

```
        after five seconds.

    def close_tooltip(self, dt):
        This function is called by the clock event described in
        display_tooltip(). When this event is executed five seconds
        after it's been registered. The tooltip widget is
        deleted from the users screen.

    def set_tooltip_text(self, text):
        We call this function and supply a string to the text
        parameter. This text relates to which ever button the users
        mouse colldied with. The text is then set and that's what's
        displayed to the user when the tooltip widget is added to
        the screen.

    def caprica_timer(self, _time):
        This function is not currently in use. It's purpose
        was to function as an independent threaded timer. The time
        was based on the number supplied to the _time parameter.
        This function ticks down until _time is == 0 displaying
        the text ...Thinking... until _time is == 0; at which
        time the text displayed is then ...Inactive...

    def start_timer_thread(self, _time):
        This function is not currently being used. But, it's
        purpose was to setup and run the caprica_timer function.

    def check_timer(self, _time):
        This function is not being used. But, it's purpose was
        to check the status of self.caprica_timer(_time). To
        ensure that it ended when _time == 0 instead of counting
        down beyond that into negative numbers.

    def get_caprica_response(self):
        This function is used to generate a response from the
        user. It combines all but the voice input/output
        responses. Basically, when you enter text into the
        user_input TextInput this function is called after
        the user hits the enter key. It then begins the
        process of the chatbot generating a response. It
        also runs as an independent thread.

    def get_caprica_voice_thread(self, words):
        This function is called when the users has activated the
        voice option, then recorded their voice. Once that
        recording process is completed this function is called.
        This function then generates the chatbots response. It
        also runs as an independent thread.

    def start_get_response_thread(self):
        We call this function after the user types some text
        into the self.ids.user_input TextInput widget, and
        then hits the enter key on their keyboard. This function
        changes the text of the notification_widget to
        '...Thinking...'. It then creates and runs the
        self.get_caprica_response() thread.

    def start_voice_response_thread(self):
        We call this function after the voice option has been
        activated, and the user has hit the record button. Once
        the record button has been clicked, the user can begin
        speaking into their microphone. Once done speaking
        we create and run the self.get_caprica_voice_thread().
        # TODO: Fix notification text.

    def _is_thread_stopped(self):
        We call this function to check if there are
        any active threads running.
        # TODO: This function is useless and should be removed.

    def _stop_threading(self):
        This function is called when an active thread is
        supposed to be terminated. The idea is that the thread
```

```
        will be interupted and thus die.
        # TODO: Remove this because it doesn't do anything.

    def get_user_text(self):
        This function is called to return the current
        text contained in the user_input TextInput widget.

    def open_delete_file_dialog(self):
        This function is called when the users clicks on the
        delete file button which is located under the settings
        submenu on the menu bar. It opens a Popup() window. Which
        contains a filebrowser and allows the user to navigate to
        the file that they wish to delete. They can then select
        the file by clicking on it, and then clicking the delete
        button on the Popup() window. Or click the cancel button
        at any time which closes the window.

    def delete_file(self, path, filename):
        This function is called after the user has slected a
        file in the Popup() window file browser and then clicked
        the delete button. The file the user selected is then
        deleted if it exists. If it doesn't exist the user is
        informed.
        # TODO: Remove path parameter as it does nothing at all.

    def delete_all(self):
        We call this function if the user clicks on the
        **Delete All** button which is located in the
        settings submenu on the menu bar. Clicking this
        button deletes all files and folders generated by the
        this program. It also then exits the program.

    def display_user_conversation(self):
        This function is called when the user clicks on
        the display conversation button. It outputs the
        contents of self.master_log into the view_port
        Widget.

    def increase_chatbot_voume(self, vol):
        This function can be called to increase the volume
        of self.engine. The volume is increased by vol. The
        values it can take are between 0-1. With 0 being the
        lowest and one being the highest. # TODO: Re-implement

    def decrease_chatbot_voume(self, vol):
        This function can be called to decrease the volume
        of self.engine. The volume is idecreased by vol. The
        values it can take are between 0-1. With 0 being the
        lowest and one being the highest. # TODO: Re-implement

    def set_volume(self, vol):
        This function is called to set the volume of
        self.engine. The volume is set to vol; vol can be
        any  value between 0-1.

    def increase_rate_of_speech(self, value):
        This funciton is called when the user increases
        the rate of speech using the settings menu. The
        current rate of self.engine is increased by value.

    def decrease_rate_of_speech(self, value):
        This funciton is called when the user decreases
        the rate of speech using the settings menu. The
        current rate of self.engine is decreased by value.

Private Members
---------------
    def __create_files(self, path)
        This function is called from within the
        self.create_dir(self, path) function
        which is called first by the
        SentienceScreen().__init__(self, **kwargs) function.
        This function when called checks to see if specific files
```

```
        exist and if they don't
        it creates them. If they do already exist if essentially
        returns none. It's also called from one other function if a
        search does not find the required files which means that
        they were intentionally or unintentionally deleted. A more
        in depth analysis of this function will be given in
        SentienceScreen().__create_files(self, path) function
        documentation.

    def __append_file(self, world, path)
        This function is caleld every time the user speaks to the
        chat bot and every time that the chat bot responds. The data
        passed to words is the response from both parties which is
        then appened to a specific file(s) which path comes from
        the path parameter. A more in depth analysis of this funciton
        will be given in
        SentienceScreen().__append_file(self, world, path) Note: The
        "World" param is a typo and needs to be changed to "word/words"

    def __set_thinking_text(self, bool):
        This function is called to change the text and the
        color of the text of the notification_widget TextInput
        to reflect the current status of the program. Ie,
        if the chatbot is about to generate a response it
        says '...Thinking...' in red text. If the chatbot has
        already generated a response it says '...Inactive...'
        in blue text.
    def __currently_thinking(self, bool):
        This function is called to determine the current
        status of the program and the chatbot. If it's
        thinking or inactive.
        # TODO: This function is redundant
Notes
-----
    This is the essential widget. It's where everything happens.
```

Definition at line 144 of file Sentience.py.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

```
def Sentience.SentienceScreen.__init__ (
              self,
              kwargs )


def __init__(self, **kwargs):

Parameters
----------
    param1 : self
Denotes this as being a member of the SentienceScree()
class.

    param2 : **kwargs
**kwargs stands for keyword arguements. This
allows an arbitrary number of keyword arguements to
be passed to the self.SentienceScreen().__init__()
function.
Attributes
----------
    mouse_pos
mouse_pos is an optional, though required for our
```

```
purposes, parameter of the Window.bind() function.
We call this function which is a member of the Window()
class. To register a mouse event. We bind the
traditional mouse_pos event to our own
self.on_mouse_pos(). The mouse (pointer) is always
tracked were simply binding it to one of our
functions so that we can monitor the position and
inctance of the pointer and call the bound function
when it's appropriate.

    self.tooltip_open
self.tooltip_open is a member of the SentienceScreen()
class. We use this as a flag to determine whether or
not the ToolTipLabel widget is being shown.

    self.mic
self.mic is a member of the SentienceScreen() class.
We use this to create our sr.Microphone() object.
This object allows us the ability to access and
manipulate the users microphone, assuming that
they have one. For later use in our program.

    self.chatbot
self.chatbot is a member of the SentienceScreen()
class. We use this to create our ChatBot() object.
We can then manipulate self.chatbot, which we do,
throughout the rest of our program. This is one of
the core objects. Without this we have no chatbot.

    self.audio_threshold
self.audio_threshold is a member of the
SentienceScreen() class. It stores an integer
value. This value enables us to force the users
microphone to ignore noises below a certain range.

    self.audio_enabled
self.audio_enabled is a member of the SentienceScreen()
class. We use this boolean variable as a flag to tell
us if the user has enabled the audio option. The user
can enable the audio option by clicking on the red
speaker button on the menu bar (Action Bar). This
sets self.audio_enabled == True and changes the color
of the icon of the speaker button to blue.

    self.audio_disabled
self.audio_disabled is a member of the SentienceScreen()
class. We use this boolean variable as a flag to tell
us if the user has disabled the audio option. The user
can disable the audio option by clicking on the blue
speaker button on the menu bar (Action Bar). This
sets self.audio_enabled == False and changes the color
of the icon of the speaker button to red.

    self.record.dynamic_energy_threshold
We use this to prevent self.record from dynamically
Ie, contantly, checking the and setting the
energy_threshold of self.record. Idealy, this should
be left as a dynamic process but because no one
microphone was created equal. Things get annoying
really fast. So I've simply set it to a static
variable for windows operating systems. And
dynamically set it once for linux operating
systems.

    self.master_log
self.master_log is a member of the SentienceScreen()
class. It's a string variable that we use to store
the users conversation with the chatbot. Every time
that the user and the chatbot say something. Their
responses are added to this string. We use this
string to write data to files.

    self.voice_enabled
```

self.voice_enabled is a member of the SentienceScreen()
class. We use this boolean variable as a flag to tell
us if the user has enabled the voice option. The user
can enable the voice option by clicking on the red
microphone button on the menu bar (Action Bar). This
sets self.voice_enabled == True and changes the color
of the icon of the microphone button to blue. It also
sets self.voice_disabled == False.

    self.voice_disabled
self.voice_disabled is a member of the SentienceScreen()
class. We use this boolean variable as a flag to tell
us if the user has disabled the voice option. The user
can disable the voice option by clicking on the blue
microphone button on the menu bar (Action Bar). This
sets self.voice_enabled == False and changes the color
of the icon of the microphone button to red.

    self.user_input
self.user_input is a member of the SentienceScreen()
class. We use this string variable to temporarily
store the contents of self.ids.user_input.text. Which
is the TextInput widget that contains the users text
comment to the chat bot. The data is returned to
self.user_input when the user enters some text and hits
the enter key on their keyboard while in the TexTInput
widget.

    self.__user_profile
self.user_profile is a member of the SentienceScreen()
class. We use this dictionary data structure to store
the users information if they choose to give it. It
stores their desired username, age, and gender. It's
not a required thing. It's optional but personalizes
a few things and helps to maintain more efficient logs
of the conversations that the chatbots has. If there
are multiple people speaking to it.

    self.username
self.username is a member of the SentienceScreen()
class. We use this string variable to store the users
desired username. Or, if the user elects not to supply
a username we give this a default value of 'User: '
and display it in the view_port TextInput widget
to display the current conversation to the user.

    self.on_mouse_pos
self.on_mouse_pos is a member of the SentienceScreen()
class. It's a function that we use to track and handle
mouse events. If the user hovers their mouse over a
button on the Menu bar (Action Bar). This function is
called, which locates teh mouses position and instance
of the mouse when it collided with a button. It then
executes the appropriate if statements which then
create a ToolTipLabel widget, change the text to
reflect the button the user collided with. And then
displays that label as a tooltip over the button.

    self.engine
self.engine is a member of the SentienceScreen()
class. We use this to create our object of the pyttsx3
class. This allows us to access the users systems text
to speech software so that the response generated by
the chatbot can be verbally delivered to the user. If
they elected to active either the audio or voice
options.

    self.record()
self.record() is a member of the SentienceScreen()
class. It's the object of the sr.Recognizer()
class. This allows us to accept, transcribe and later
manipulate an audio recording of the user. This
occurs when the user has activated the voice option.

```
    self.__is_thinking
self.__is_thinking is a member of the SentienceScreen()
class. We use this boolean variable as a flag to tell
us whether or not the chatbot is preparing to generate
a response for the user. Or has just finished generating
a response to the user. When the chatbot is generating
a response the text of the notification_widget is set
to '...Thinking...' and the color of that text is red.
When the chatbot finishes generating a response and has
sent it to the user the text is set to '...Inactive...'
and is blue.

    self.current_conversation
This is a member of the SentienceScreen() class.
We use this string variable to store the current
contents of the view_port Widget. When a tooltip
is displayed. We do this to prevent the loss of
the information that was previously being displayed.
Members
-------
    super(SentienceScreen, self).__init__(**kwargs)
Here we're calling super dynamically to allow the
use of inheritance. This applies to the
sentienceScreenManager() class. It allows us to
work with the various widgets and screens.

    Window.bind(mouse_pos = self.on_mouse_pos)
We call Window.bind() to bind the base Window
classes mouse_pos event to our mouse event. Which
in this case is self.on_mouse_pos()

    threading.Event()
threading.Event() is a member of the threading()
class. We use this to create a threading event
which we'll use to interupt active threads later on.

    Factory.ToolTipLabel(text = (string))
We use this to register and instantiate
classes anywhere anytime. In our case though
we're just setting this up and setting the text
field to '', Ie, an empty string.

    Config.set('input', 'mouse', 'mouse', disable_multitouch)
Config is a member of the kivy base class. We call
this in our SentienceScreen.__init__() method
to disable kivys multitouch ability. This shuts off
users ability to interact via touch screen on touch
screen capable systems.

    sys.platform.startswith(string)
This is a member of the sys() class. We call this
function to dertmine whether what operating system
the user is using. It returns a boolean value, if
the version matches either 'linux' or 'win'.

    pyttsx3.init(string)
This is a member of the pyttsx3() class. We call
this function when we declare and instantiate
our object of this class. It also serves to
set the driver for the systems text to speech
software based on the users operating system.

    sr.Recognizer()
This is a member of the speech_recognition() class.
We call this when we declare and instance our
self.record object. Which then allows us to
accept user input from a microphone and then
transcribe that uadio respone as a string for
later manipulation.

    sr.Microphone()
This is a member of the speech_recognition() class.
```

```
We call this when we declare and instantiate our
self.mic object. Which then allows us to manipulate
the users microphone if they have one.

    ChatBot()
Here we setup the ChatBot. We do wo when we decalre
and instantiate our self.chatbot object. We create and
supply the required filters and adapters which dictate
how this chatbot will learn.

    self.set_gender()
This is a member of the SentienceScreen() class. We
call this function to set the gener of self.engine
to a female. This has the effect of changing the
default voice from a male, to female voice.

    self.set_speech_rate()
This is a member of the SentienceScreen() class. We
call this function to set the speech rate of the
users systems speech to text software. In our
case we lower it so that when self.caprica_speak()
is called the resulting spoken string is done
so in a manner that the user can understand.

    leng()
We call the built in python leng() or length
function to dertmine the length of self.username.
If the length is less than or equal to zero we
supply self.username with the default value of
'User: '. If the user elects later on to set their
own username then the self.user_profile overrides
this variable.

    self.create_dir(path)
This is a member of the SentienceScreen() class.
We call this function to create a series of files
and folders that the user needs to operate
this program.

    self.engine.connect(string, event)
We call this function to bind our events
to the pyttsx3 events. We connect self.onEnd
to the pyttsx3 'finished-utterance' event. This
event is fired when the pyttsx3 finishes speaking
whatever string was supplied to it. We also connect
self.caprica_speak to 'started-utterance' which is
fired when the systems text to speech software
begins speaking a supplied string.
Private Members
---------------
    None

Exceptions
----------
    None

Returns
-------
    None

Notes
-----
    This is the initalization method of SentienceScreen().
    It's relatively comprehensive so I'm not going to explain
    it again. It's easy enough to understand whats happening
    when you reference the above comments.
```

Definition at line 662 of file Sentience.py.

Here is the call graph for this function:

6.5.3 Member Function Documentation

6.5.3.1 __append_file()

```
def Sentience.SentienceScreen.__append_file (
              self,
              words,
              path )  [private]
```

```
def __append_file(self, words, path)
    This function is called every time the user
    and or the chat bot speaks. It Appends every
    every conversation to the appropriate file.


Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

    param2 : words
Words is a string variable that contains
the response spoken by either the chat bot
or the user. This is the string that's appended
to the appropriate text file.

    param3 : path
The path variable is passed to this function
from the self.create_dir(self, path) function
which is also the calling function for
self.__create_files(self, path).


Attributes
----------
    path
The path variable here is a reference
to the absolute file path of a specific
file. This function is called every time a
response is entered by the user and generated
by the chat bot. The response are then appended
to User_Statements, Caprica_Statements respectively.

Members
-------
    os.path.isfile('path to file')
We call this function to ensure that
the files we're attempting to manipulate
already exist. If os.path.isfile() == True
then the file exists and the data stored in
the words variable is appended to the end of
the file. If os.path.isfile() == False then
the file does not exist and we re-call the
function self.__create_files(self, path).

Private Members
---------------
   self.__create_files(self, path)
       This function is called only if one
       of the files required files has been deleted.
       This function will then write the file to
       the disk.


Returns
```

```
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error log
text file.

    FileNotFoundError
This can occur in a variety of ways however my primary
concern is that file path the user selected is broken.
Resulting in an File Not Found error. If this occurs
it's handled and logged to an error file text log.

    NameError
Again this can occur in a variety of ways but the
primary concern is that the conversion to bytes does
not take place or breaks some how due to wacky Unicode
characters. In which case the exception is handled and
logged to an error log text file.
Notes
-----
    This function is called every time the user or
    the chat bot inputs/generates a response. That response
    is then appended to it's respective file.

    1: User response: User_Statements.txt

    2: Chat bot response: Caprica_Statements
```

Definition at line 3630 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.2 __create_files()

```
def Sentience.SentienceScreen.__create_files (
              self,
              path )  [private]


def __create_files(self, path)
    This function is called during the
```

```
    SentienceScreen().__init__() function.
    From within the self.create_dir(self, path)
    function.

Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

    param2 : path
The path variable is passed to this function
from the self.create_dir(self, path) function
which is also the calling function for
self.__create_files(self, path).

Attributes
----------
    path
The path variable stores the path to the
location where we previously created a
folder. This is the same path that we will
use to create three text files.
Caprica_Statements.txt
User_Statements.txt
Error Logs.txt
We simply append those three file names to the
end of the passed path variable.

Members
-------
    os.path.isfile('path to file')
We call this function to ensure that
the files we're attempting to create
don't already exist.
If os.path.isfile() == True then the
file(s) exist and we do nothing.
If os.path.isfile() == False then the
files do not exist and we create them.

Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.
```

```
    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error
log text file.

    FileNotFoundError
This can occur in a variety of ways however my primary
concern is that file path the user selected is broken.
Resulting in an File Not Found error. If this occurs
it's handled and logged to an error file text log.

    NameError
Again this can occur in a variety of ways but the
primary concern is that the conversion to bytes does
not take place or breaks some how due to wacky Unicode
characters. In which case the exception is handled and
logged to an error log text file.
Notes
-----
    This function is called during the initialization of
    SentienceScreen() from within the
    self.create_dir(self, path) function. The purpose of
    self.__create_files(self, path) is to create  series of
    files which we will use to store.

    1: Caprica_Statements : Stores all response from the
                    chat bot.

    2: User_Statements : Stores all responses from the user.

    3: Error Logs : Stores any exceptions that occur with a
            time date and calling function stamp.

    4: Username + _Conversation : This file will be eventually
              created and stored to maintain a
              comprehensive list of all chat bot and user
              responses as they relate to each other.
```

Definition at line 3454 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.3 __currently_thinking()

```
def Sentience.SentienceScreen.__currently_thinking (
              self,
              bool )  [private]


__currently_thinking(self, bool)

param1: self
    Denotes this as being a member of the SentienceScreen()
    class.

param2 : bool
    A boolean variable is passed to this function and
    then used to dertime the text in the self.notification_widget
    TextInput Widget.

Attributes
----------
    bool
bool is used to store the boolean variable
passed to this function upon its function call.
Members
-------
    None
```

```
Privat Members
--------------
    self.__is_thinking
self.__is_thinking is a member of the SentienceScreen()
class. We use this to monitor the status the 'thinking'
status of the program and to switch it on and off. If
it's on self.notification_widget has its text field set
to '...Thinking...' with a red foreground. If it's off
self.notification_widget has its text field set to
'...Inactive...' with a blue foregrounf.
    self.__set_thinking_text(bool)
Once we know if the program is currently thinking
we call this function which is a member of the
SentienceScreen() class. To actually change
the text field of the self.notification_widget.
Notes
-----
    We call this function to check to see if the chatbot
    is about to begin generating a response to the user.

    If the chatbot is about to begin it's generation process
    we set the self.__is_thinking variable to True, we then
    call self.__set_thinking_text(bool) to change the
    text field of self.notification_widget to '...Thinking...'
    with a red foreground.

    If the chatbot is done generating a resposne we set
    the variable self.__is_thinking to False, and then call
    self.__set_thinking_text(bool) to change the text field
    of the self.notification_widget to '...Inactive...' with
    a blue foreground color.
```

Definition at line 6114 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.4 __set_thinking_text()

```
def Sentience.SentienceScreen.__set_thinking_text (
              self,
              bool )  [private]


__Set_thinking_text(self, bool)

Parameters
----------
    param1 : self
Denotes this as being a member of the SentienceScreen()
class.

    param2 : bool
The boolean variable contains the dertmining factor
for how the text will be set and what color that text
will be in the self.notification_widget. It recieves
this information from the
self.__currently_thinking() function.
Attributes
----------
    bool
if bool == True then the chatbot is about to begin
generating a response for the user. We then set the
text field of self.notification_widget to
'...Thinking...' and set the foreground to red. If
it's False we know then that the chatbot just
finished generating a response. We then set the text
field of self.notification_widget to '...Inactive...'
with a blue foreground.
Members
```

```
-------
self.notification_widget.text
    We use this to set the text field of the
    self.notification_widget TextInput widget to the
    appropriate text based on the preceeding conditions. This
    is a member of the SentienceScreen() class.

self.ids.notification_widget.foreground_color
    We use this to give the text a color. The color is relative
    to the programs current status. If the program is
    '...Thinking...' then the text will be red. If the program
    is '...Inactive...' then the text will be blue. This
    function is a member of the SentienceScreen() class.

kivy.utils.get_color_from_hex('hex string')
    This is a member of the Kivy base class. We use this
    function to convert a hexadecimal string to a compatiable
    color read as an integer by the self.ids.notification_widget
    TextInput widget.
Private Members
---------------
    None
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
    We call this function to change the self.notification_widget
    TextInput widgets text field to either '...Thinking...' or
    '...Inactive...'. This status is determined when the chatbot
    is generating or finished generating a response. If the
    chatbot is generating a response it's thinking. If it has
    finished generating a reponse then it's set to inactive.

    We then change the color of the text using a hex string
    which is then read as an integer byt the
    self.notification_widget.foreground_color property.
    If the program is thinking then the text is made red.

    If the program is inactive then the text is made blue.
```

Definition at line 6033 of file Sentience.py.

Here is the caller graph for this function:

6.5.3.5 caprica_speak()

```
def Sentience.SentienceScreen.caprica_speak (
              self,
              words )


def caprica_speak(self, words)

Parameters:
-----------
    param1 : self
Denotes it as being a member of
SentienceScreen(Screen) class.

    param2 : words
The parameter words contains the chat bots
response to the user.


Attributes
```

```
----------
    self.ids.user_input
The string contained in the user_input TextInput
Widget is cleared and the hint_text is reset.



Members
-------
    self.onEnd(self)
This function is called every time that the
self.caprica_speak() function has been called. Once
self.caprica_speak() finishes speaking the passed string.
self.onEnd() is fired because it's bound to the
'finished-utterance' event. This ends the speaking
loop and empties the event queue.

    self.engine.say(str(words))
This function is called from within the
self.caprica_speak() function. this is the function
which access the systems tts software and actually
verbally 'speaks' the string passed to it.

    self.engine.startLoop()
This function is called to ensure that the string passed
to self.caprica_speak() is fully spoken. Ie, it ensures
that the entire string is read before the event
'finished-utterance' is fired.

Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error
log text file.
Notes
-----
    We call this function if self.auido_enabled == True
    and or if self.voice_enabled == True. The purpose
    of this function is to access the programs tts
```

```
    software to verbally speak the string passed to it.

    We start off by checking the users operating system.

    If sys.platform.startswith('linux') == True
    the user is using a Linux based operating system and
    the appropriate if statements are executed.

    Otherwise if sys.platform.startswith('win') == True
    then the user is running a windows based operating
    system and the appropriate if statements are executed.

    We then sen a string to self.engine.say() which access
    the systems tts software and reads the string it's sent.
    Which is in this case the response of the chat bot.

    self.engine.say(str(words))
    We then call self.engine.startLoop() to start a loop
    ensuring that the string(s) sent to self.caprica_speak()
    are all read.

    Finally, we clear the user_input TextInput Widget
    resetting its hint_text property as well.
```

Definition at line 2080 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.6 caprica_timer()

```
def Sentience.SentienceScreen.caprica_timer (
              self,
              _time )


def caprica_timer(self, _time)

Parameters
----------
    param1 : self
Denotes this as being a member of the SentienceScreen()
class.
    param2 : _time
Can be either double or of type int. I'm using it
as an integer by supplying it with a whole part. This
variable dictates how long the timer which is this
function runs.
Attributes
----------
    mins
This variable stores the number of minutes that
this timer function will run. mins is displayed and
along with secs ticks down to reflect the amount of
time that this function will run. Though the user can't
see the visual display.
    secs
This variable the number of seconds that this timer
function will run. secs is displayed and along with
mins ticks down to reflect the amount of time that
this function will run. Though the user can't see
the visual display.
    timeformat
timeformat is the format of how the time will apear
to the user when it's printed to the console. It
looks like this. If you supply, 168 to this function
it would output 2:48. Though the user can't see this
visual timer.
Members
-------
```

```
    time.sleep(integer)
We call time.sleep() to ensure that the timer
only counts down 1 second at a time and that it
doesn't interfere with any other active thread.
    divmod()
This is a builtin python function which returns
the quotient and remainder of the two numbrs
whic are supplied to it; in this case mins, secs.
    str().format()
This is a member of the built in python string class.
It formats teh string to look however you set it. In
our case we format the ticker display to print out
2:48 if supplied with 168, if it were 120 it would
look like 2:00.
    self.check_timer(_time)
This is a member of the SentienceScreen() class.
We call this function to ensure that _time is
not less than or equal to zero if it is we terminate
both self.check_timer and self.caprica_timer().
    self.notification_widget.foreground_color
This is a member of the SentienceScreen() class. It's
one of our TextInput widgets. We use this to change the
foreground color, which is the color of the text. To
reflect the active status of the program. If the chatbot
is about to generate a response for the user the color
of the text is changed to red. If the chatbot has just
finished generating a response to the user the color
of the text is blue.
    self.notification_widget.text
This is a member of the SentienceScreen() class. We
use this to set the text property of the
self.notification_widget which is one of our
TextInput widgets. We do this to reflect the current
status of the program. If the chatbot is about to
generate a response for the user we change the text
to '...Thinking...' and set the color of the text to
red. If the chatbot has just finished generating a
response to the user we set the text to '...Inactive...'
and change the color of the text to blue.
    kivy.utils.get_color_from_hex()
This is a member of the kivy.utils() class. We call
this function to convert a hexadecimal string to
an integer or double value that (automaticaly double)
that can be interpereted by the TextInput widget
as an appropriate and existing color code. Kivy uses
the opengl method setting colors and it's easier
for me to work with hex then it is for me to determine
the rgba-opnegl equivelant.
Private Members
---------------
    None
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
    This function is not currently being used. But,
    an explanation of it's use is as follows. The developer
    supplies a number to the _time variable. This number
    represents the time that this function will run.

    This function should run as an independent thread which
    constantly ticks down until _time = 0. While it ticks
    down it should also flash the text '...Thinking...'
    until the function terminates when it sets the text
    to '...Inactive...'
```

Definition at line 4458 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.7 check_timer()

```
def Sentience.SentienceScreen.check_timer (
              self,
              _time )


check_timer(self, _time)

Parameters
----------
    param1 : self
self denotes this function as being a member of
SentienceScreen().

    param2 : _time
_time is a double variable which contains a number.
This number is used in self.caprica_timer as a countdown.
This function monitors that countdown and ensures that
when time is <= 0 the while loop in self.caprica_timer
is broken. We also use this to know when to
disable/enable some other features in that function.
More information about self.caprica_timer can be found
in it's comments. For our purpose here we check _time
to see if it's <= 0 if it is we return True if
_time is > 0 we return False.
Attributes
----------
    _time
See above information in Paraemters section.
Members
-------
    None
Private Members
---------------
    None
Exceptions
----------
    None
Returns
-------
    True
We return True if _time is <= 0.
    False
We return False if _time is > 0.
Notes:
    This function is called during self.caprica_timer
    to check the variable _time. If the number stored in
    the variable _time is less than or equal to 0 we
    return True. If the number stored in _time is
    greater than 0 we return False.
```

Definition at line 4671 of file Sentience.py.

Here is the caller graph for this function:

6.5.3.8 clear_viewport()

```
def Sentience.SentienceScreen.clear_viewport (
              self )


def clear_viewport(self):
    We call this function a few times. It's
    probably the simplest one here to understand.
    It's purpose is only to reset the text in the
    view_port TextInput Widget to an empty '' string.
```

```
Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.



Attributes
----------
    self.ids.view_port
view_port TextInput Widget is one of our main TextInput Widgets
which displays the text conversations between the user and the
chat bot.

Members
-------
    None

Private Members
---------------
    None

Returns
-------
    return None

 Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure that
the chat bot doesn't die. Essentially I just need to make
sure that it completes and executes the python text to speech
functions in a manner that doesn't cause a fatal exception. If
something does occur the exception will be handled and logged to
an error log text file.

    ValueError
Ensures that values passed to the chat bot are appropriate.
And if for some reason one isn't the exception will be handled
and logged to an error log text file.

Notes
-----
    The purpose of this function is only to reset the text
    in the view_port TextInput Widget to an empty '' string.
    Which also has the effect of resetting the hint_text
    property.
```

Definition at line 2345 of file Sentience.py.

6.5.3.9 close_tooltip()

```
def Sentience.SentienceScreen.close_tooltip (
              self,
              dt )


close_tooltip(self, dt)

Parameters
----------
    param1 : self
self denotes that this is a member of SentienceScreen().

    param2 : dt
The dt parameter is a float (double) value. It refers
to a time. So in our case we supply the number 5 to
this parameter when this function is called in
self.display_tooltip(). The number 5 refers to
milliseconds/seconds/frames. The time at which
this function is called will be different from
system to system but will not exceed 5 seconds.

Attributes
----------
    self.tooltip
tooltip is a reference to the ToolTipLabel Widget
in the kv design language. This the instantiated and
mutable object of that widget.

    self.tooltip_open
We use tooltip_open to check whether or not the tooltip
widget is currently "open", in other words, in use. If
tooltip_open == True then we know that the tooltip
widget is currently in use and we can close it. If
it's False we know it's not in use and that it can
be opened to display information about a widget on
the menu bar (ActionBar).
    Window
The window member relates to the kivy Window.
The Window is the main active root widget.
This should not be confused with root_widget.
The root widget that Window refers to is the
windowing system its self which is default and
separate from any user generated widgets.
Members
--------
    Window.remove_widget(self.tooltip)
This allows us to remove (delete) a widget
from the current active window (widget). Remember
this refers to the windowing system and the main
window. That is to say that we can use this to
remove a user created widget from the MainWindow.
In this case we use it to remove SentienceScreen.tooltip.
self.tooltip is the only parameter supplied to this function
call as it's the only widget that we remove.

Private Members
---------------
    None
Returns
-------
    None
Exceptions
----------
    None
Notes
-----
    I've outlined what this function does fairly well in
    the above comments. But, an overview of the function
    is this.
```

```
    We call self.close_tooltip(event, dt) with the kivy
    Clock.schedule_once(event, dt) function. We only
    call self.close_tooltip if self.tooltip_open == True.

    Calling this function allows us to remove the tooltip
    (Label) with descriptive text from the screen.
```

Definition at line 4311 of file Sentience.py.

Here is the caller graph for this function:

6.5.3.10 create_dir()

```
def Sentience.SentienceScreen.create_dir (
              self,
              path )


def create_dir(self, path)
    This function is called during the
    SentienceScreen().__init__() function.
    It creates a directory (folder) that
    will be used to store a series of files
    in.


Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

    param2 : path
This is the path for the folder we're about
to create in the function self.create_dir(path)


Attributes
----------
    path
The path variable stores the path to the
location where we will create a folder on
the users operating system. We will create
a series of required files when we call the
self.__create_files(path) function.
This path is based on the users operating system.


    Members
    -------
os.mkdir()
    This function is called to access the systems native
    directory creation process. On linux the command is
    simply mkdir. Whereas on windows you're accessing
    the win32 api and calling the C CREATE_DIRECTORY
    binding function.

Private Members
---------------
    self.__create_files(path)
We call this function after we've created the
folder that we intend to store the required files in.
We pass one parameter to this function and it's path.
I've set the files names to be specific so All I need
to do is path + 'file name' inside the function
self.__create_files(path)

Returns
-------
```

```
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error log
text file.

    FileNotFoundError
This can occur in a variety of ways however my primary
concern is that file path the user selected is broken.
Resulting in an File Not Found error. If this occurs
it's handled and logged to an error file text log.

    NameError
Again this can occur in a variety of ways but the
primary concern is that the conversion to bytes does not
take place or breaks some how due to wacky Unicode
characters. In which case the exception is handled and
logged to an error log text file.
Notes
-----
    This function is called during the initialization of
    SentienceScreen() it's purpose is to create a folder.

    In this folder we store a number of text files which
    are created after the folder has been made; at which time
    another function is called from within self.create_dir(path)
    which then creates those text files.
```

Definition at line 3306 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.11 create_user_profile()

```
def Sentience.SentienceScreen.create_user_profile (
              self )


def create_user_profile(self)
    We call this function after the user has entered
    his/her desired username. It's only purpose is to
    check if the length of the string that stores the
```

```
    username is greater than 0. If it's not it sets a
    default value to the username of 'User: '.
    If it is greater than zero this function calls the
    self.caprica_speak(string) function and says
    'Hello ' + self.username

Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

Attributes
----------
    self.username
This is the string variable that contains the users
desired input username. If no username is set a
default value of 'User: ' is set to self.username

Members
-------
    self.caprica_speak(string)
We call this function with the string
'Hello' + self.username Essentially the call to
self.caprica_speak(string) from the function
self.create_user_profile(self) is just a way to
personalize the experience and deliver a verbal greeting
to the user.

Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error log
text file.
Notes
-----
    This functions purpose is just a way to personalize the
    experience and deliver a verbal greeting to the user.

    As always we start off with a system check.
```

```
    If sys.startswith('linux') == True then we know
    that the user is using a Linux based operating system
    and the appropriate if statement is executed.

    Otherwise if sys.platform.startswith('win') == True
    we know the user is using a windows based operating system
    and the appropriate if statements are executed.

    We then check the length of self.username
    len(self.username) <= 0 if this is True
    we know that the user did not set a username
    and we set a default value for self.username of
    'User: '.

    If len(self.username) > 0 we know that the user has entered
    as username and we call
    self.caprica_speak('Hello' + self.username) to deliver a
    personal greeting to the user.
```

Definition at line 2423 of file Sentience.py.

Here is the call graph for this function:

6.5.3.12 decrease_chatbot_volume()

```
def Sentience.SentienceScreen.decrease_chatbot_volume (
              self,
              vol )
```

```
    Decreases the chatbots volume by vol.
```

Definition at line 1043 of file Sentience.py.

6.5.3.13 decrease_rate_of_speech()

```
def Sentience.SentienceScreen.decrease_rate_of_speech (
              self,
              value )
```

```
    Decreases the chatbots words spoken per minute.
```

Definition at line 1222 of file Sentience.py.

6.5.3.14 delete_all()

```
def Sentience.SentienceScreen.delete_all (
              self )


delete_all(self)

Paraemters
----------
    param1 : self
Denotes this as being a member of
SentienceScreen().
Attributes
----------
    temp
temp is a local variable which we use to store the
absolute directory path; to the folder that was
created when this program was first run. This folder
varies based on the users operating system. We pass
this variable to our deletion tool.
 ignore_errors
     This is sort of a misnomer. This a member of the
     shutil.rmtree() class. It doesn't actually ignore
     any errors. It just forces the function to delete
     the files and or folders and more importantly, to
     not print out any errors. It's a boolean variable
     and we set it to true to ensure that it does in fact
     force the deletion of the folder.
Members
-------
    sys.platform.startswith('string')
We use this function to detect the users operating
system. It will determine whether or not you're using
a windows or linux based operating system. It doesn't
search for a specific version. It just ensures that
you're using one of them. This allows us to make this
program cross platform.
    shutil.rmtree(path, optional_boolean)
       We use this function to preform our deletion operations.
       By default it makes use of the native systems api to
       preform this operation. We need to suuply it a path,
       this where our temp variable comes in. You'll remember
       that we stored the path to the directory in it. We
       also suply this function with a boolean variable
       ignore_errors and set it to True. As mentioned above
       this simply ensures that the folder gets deleted,
       whether it's empty (this is what it checks for) or not
       and prevents it from spitting out any warnings or errors
       from the system. This bool vairable is by default set
       to False.
    self.ids.view_port.text
This is the view_port TextInput widget in the
kv code. We use this widget to display certain
warnings and conversations to the user. In this
case when the folder has been deleted we inform the
user by printing out the folders path and stating
that it has been deleted.
    Private members
    ---------------
self.__append_file(string, filepath)
   This function is a member of SentienceScreen(). We
   call this function to append error messages if they
   occur to an error log. We supply the function,
   the exception, what occured, and the date and time
   that it occured to this fileself.
    Returns
    -------
None
    Exceptions
    ----------
IOError
```

```
    The IOError can occur due to many reasons.
    My primary concern is file manipulation. The
    improper opening/closing/writing to files. If
    the exception occurs it's handled and logged; in
    an error log text file.
FileNotFoundError
    This can occur in a variety of ways however my
    primary concern is that file path the user selected
    is broken. Resulting in an File Not Found error.
    If this occurs it's handled and logged to an error
     file text log.
    Notes
    -----
First we check to make sure the user is running a
viable operating system. Rather, one that's
compatible with this program. We then execute the
appropriate code.

We create a variable named temp and sore the absolute
path of the directory (folder) that we're going to
delete.

We then call shutil.rmtree() to make access of the
systems native api to delete the directory. Once deleted
we display a message which includes the full directory
path and a string stating that the folder has been
deleted.

If any errors occur we record and log them to an error
log.
```

Definition at line 5908 of file Sentience.py.

Here is the call graph for this function:

6.5.3.15 delete_file()

```
def Sentience.SentienceScreen.delete_file (
              self,
              path,
              filename )


delete_file(self, path, filename)

Parameters
----------
    param1 : self
Denotes this as being a member of the SentienceScreen()
class.

    param2 : path
contains the partial path to the file
returned to it by the users selection in the
DeleteDialog() pop up window. This parameter
does not contain the filename. Nor is it ever
used. It's pointless to even be here.

    param3: filename
filename contains the full file path to the
file that the user selected for deletion by
clicking on the file and then clicking the
delete button in the DeleteDialog() Popup()
window.
Attributes
----------
    temp
temp contains teh formatted and fbsolute filepath
to the file that the user selected for deletion. This
```

```
path is then passed to os.remove(temp) to carry out
the actual deletion process.
Members
-------
    os.path.isfile(path)
We call this function which is a member of the
os() class. To ensure that the filepath passed to
it does indeed exist. If it does exist this function
returns tru and the appropriate if statement is
executed. If it returns False the nthe file does not
exist and again the appropriate if statement is
executed.

os.remove(path)
    We call this function which is a member of the os()
    class to carry out the deletion process of the file
    that the user selected. This call makes use of the
    users systems native API deletion feature.

self.ids.view_port.text
    We call this function which is a member of the
    SentienceScreen() class. To change the text of the
    view_port TextInput widget. The text set depends on
    whether or not the file was deleted. If the file
    selected did exist and was deleted the text is
    changed to 'Filepath File has been deleted'. If
    the file did not exist and therefor was not deleted
    then the text is set to 'Filepath either does not exist
    or was already deleted'.
Private Members
---------------
    None
Exceptions
----------
    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    FileNotFoundError
This can occur in a variety of ways however my primary
concern is that file path the user selected is broken.
Resulting in an File Not Found error. If this occurs
it's handled and logged to an error file text log.
Returns
-------
    None
Notes
-----
    This function is called after the user selects a file
    that they wish to delete in the DeleteDialog() Popup()
    window. Then proceeds by clicking the delete button on
    that Popup() window. The partial path with out the
    file name is returned to this funtion but is not used.

    The full filepath is also returned and stored in the
    variable filename. We store filename in the string variable
    temp. We then strip the first two, and last two characters
    in the temp variable. Filename is returned as tupple and
    so it contains '[/example/filepath/random.text]'.

    We then ensure that the selected file does exist
    and if it does we delete it and inform the user
    that the file was succesfully deleted. If the file
    doesn't exist during the initial check we then
    inform the user that it doesn't exist or that
    it's already been deleted.
```

Definition at line 5786 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.16 dismiss_popup()

```
def Sentience.SentienceScreen.dismiss_popup (
              self )


def dismiss_popup(self): is called when the user clicks
the Cancel button on the Popup

self._popup.dismiss() : calls the built in to dismiss the Popup
```

Definition at line 3994 of file Sentience.py.

Here is the caller graph for this function:

6.5.3.17 display_tooltip()

```
def Sentience.SentienceScreen.display_tooltip (
              self,
              args )


display_tooltip(self, *args):

Parameters
----------
    param1 : self
Denotes this function as being a member of
self.SentienceScreen().

    param2 : *args
Can take a list, array dict, etc.. of
arguments. This relates to the specific position and
instance of the pointer (mouse) when this function is
called.
Attributes
----------
    self.tooltip
self.tooltip is the widget ToolTipLabel declared in
the kv design language. This is the tooltip widget
that we use to display the text which describes the
buttons the user hovers over.
Members
-------
    Window
The window member relates to the kivy Window.
The Window is the main active root widget.
This should not be confused with root_widget.
The root widget that Window refers to is the
windowing system its self which is default and
separate from any user generated widgets.

    Window.add_widget()
When this is called we add a new widget
to the main active window. In this case
we're adding a Label widget which contains
descriptive text about the specific button
that the user is hovering over when this function
is called.

    Clock
This is the kivy clock, not the system clock.
This handles all of the frames, callbacks and events
in a kivy program. That is to say that this is what makes
everything work in that it calls things rhythmically and
prevents any thing from occurring concurrently witch could
break the program. It also has other uses such as registering
function calls that will occur at or during specific intervals.
```

```
    Clock.schedule_once(event, time)
       Clock.schedule_once() is a way for us to
       call a specific function once (not recursively,
       or repetitively). This function call requires
       an event, such as the calling of a function, and
       a time frame, this time frame dictates when the
       event occurs. In our case we call the event
       five seconds after it's been registered here.
       Or to be more accurate we call it five frames
       after. Due to the way the kivy clock functions
       the amount of time that this is executed in will
       not always occur at the same time for a variety of
       reasons. In actuality on most systems the call
       will occur around .5 seconds after the event has been
       registered. This function is used to call the
       SentienceScreen().close_tooltip() function which
       removes the tooltip from the screen.

    close_tooltip()
self.close_tooltip is a member of SentienceScreen().
We call this function to remove the tooltip from the
screen. It's called with the clock event.
Private Members
---------------
    None
Returns
-------
    None
Exceptions
----------
    None
Notes
-----
    I've outlined what this function does
    fairly well in the above comments. But,
    an overview is this. This function is called
    when the user hovers their mouse over a button
    on the menu bar (ActionBar). That specific
    instance of the pointer (mouse) is then passed
    to *args. We then add the ToolTipLabel Widget
    to that button after the specified amount of
    time.
```

Definition at line 4214 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.18 display_user_conversation()

```
def Sentience.SentienceScreen.display_user_conversation (
              self )
```

```
    When called this function displays the contents of
    self.master_log inside the view_port TextInput Widget.
```

Definition at line 1016 of file Sentience.py.

6.5.3.19 get_caprica_response()

```
def Sentience.SentienceScreen.get_caprica_response (
              self )
```

```
get_caprica_response(self)

Attributes
----------
    my_timer
my_timer is the decleration and intialization of
threading.Thread(target = event, args = (params)).start()
This one line code creates and starts a new thread. This
thread allows us to use the self.caprica_timer() function.
    target
This variable is a member of the threading.Thread()
class. It's used to register the passed event. Then
call said event which is in this case the function
self.start_timer_thread; which in turn calls the
function self.caprica_timer.
    args
This variable is a member of the threading.Thread()
class. It's used to store the parameters of the
event thats passed to the target member of the
threading.Thread() class. In this case we pass
a number to it. This number is a double variable
and refers to the amount of time that will be used
in the self.caprica_timer function.
    response
The response variable is used to store the chatbots
response. It's that simple. We call the chatbots
function to get the response by passing it the users
statement/question, etc.. The chatbot then searches the
database for a response which bests fits teh string
passed to the chatbots function. The returned data is
then stored in the temp variable for later use and
manipulation. This variable is used through out
self.get_caprica_response() function except when
the user has enabled the voice option and makes use
of the voice option.
    self.master_log
This is a string variable which as its name states
contains a master log of the conversation. In other
words, it stores both the users text and the chatbots
text in order as it's entered. This is done so that we
can write a full file of the entire conversation. As it
occurs. This is not done real time. It's done when the
user clicks the "Write logs button" which is represented
by a pencil on the menu bar (Action Bar).
    self.username
self.username contains the users username. This assumes
that the user created a username. If the user did not
create a username then a default value of 'User: '
is provided. This is used in various ways: We set
the view_port TextInput Widget conversation log
with User: my statement. We append this data to the
self.master_log string. We append this data to the
User_statement.txt file.
    self.audio_disabled
This is a boolean variable which we use to check whether
or not the user has disabled the audio option. If
self.audio_disabled == True; then the audio option is
disabled and the chatbot can only communicate with the
user via text. If self.audio_disabled == False then
self.audio_enabled == True; meaning that the audio mode
is enabled and the chatbot can access the systems
text to speech software and communicate verbally with
the user.
    self.audio_enabled
This is a boolean variable which we use to check
to see if the user has enabeld teh audio option.
```

```
If the user has enabled the Audio option then Caprica
can access the systems text to speech software and
speak directly to the user.
If self.audio_enabled == True then the user has
enabled the Audio option; and the chatbot can then
speak to the user verbally.
If self.audio_enabled == False; then the user has either
disabled the audio option or not bothered to enable it
yet which means that the chatbot can only communicate
with the usr via text.
    self.voice_enabled
This is a boolean variable which we use to check
to see if the user has enabled the voice option.
The voice option enables the user to access and use
their microphone to speak directly to Caprica. If
the user doesn't have a microphone then they can't use
this option. If self.voice_enabled == True the user has
turned on the voice option and does have a microphone.
If self.voice_disabled == False; the user has either
disabled the voice option or doesn't have a microphone.
    temp
The temp variable is used to store the chatbots
response. It's that simple. We call the chatbots
function to get the response by passing it the users
statement/question, etc.. The chatbot then searches the
database for a response which bests fits teh string
passed to the chatbots function. The returned data is
then stored in the temp variable for later use and
manipulation. This variable is only used when the user
has activated the voice option.
    self.mic
self.mic is the initialized object of the
SpeechRecognition.Microphone() class. With
this object we can access the users microphone
and listen to the audio then pass into the
recognizer object for transcription; and later
storage as a string.
    source
The source variable is created to pipe the audio opened
by self.mic; into the variable audio (which is an
audio file). Source doesn't store the data. It simply
passes the data into the audio variable as it's picked
up by the users microphone. This of course assumes that
the user has a microphone. If the user doesn't have a
microphone then the user wont ever get into this
function. The source object is cleared and destroyed
when the with loop is ended. Using the with loop
functionality automatically closes the loop, clears
teh data, and deletes the object when the considiton
reaches its breakpoint. In this case the breakpoint is
when the user stops speaking. So basically, while this
microphone is picking up noise pipe it through source
and store it in the audio variable. When it stops picking
up noise end the loop and clean up the data.
    statement
This is a string variable which is used to store
the string returned by the
self.record.recognize_sphinx(audio) function. The
audio file passed to the above mentioned function is
transcribed and returned as a string to the statement
variable.
Members
-------
    self.start_timer_thread(self, _time)
This function is called when it's passed to the
my_thread object. When the new thread is started
this is used to call the self.caprica_timer()
function. The double variable passed to it
which represents the amount of time that
self.caprica_timer() runs.
    self.chatbot..get_response(words)
This function looks exeedingly complicated.
But, it's not. Simply put this function is
```

```
what takes the input from the user_input
TextInput widget; passes it to the chatbot
so it can locate an appropriate response by
searching its database and then returns that
string to either another variable or to a function.
That string is then communicated to the user as
the chatbots response. It checks to see if the
user has enabled or disabled the audio and or
voice modes. From there it accepts the input as
it's intended to.
    threading.Thread(target = (), args = ())
This creates a new thread, this thread
refers to a function or other event
and the parameters of arguements to be
passed to that event. So in our case,
we use this to start a timer which counts
down from the number supplied to args; the
function self.caprica_timer then preforms
the count down which is checked by
self.check_timer to ensure that the double
variable stored in _time is less than
or equal to zero. If it's equal to zero the
function ends. If it's not equal to zero
the function display the text '..Thinking..'
in the notification_widget TextInput widget.
    threading.Thread().start()
This function simply starts the new thread
that was created. That is to say this function
starts the my_thread thread.
    self.get_user_text()
This function is used to return the text
contained in self.ids.user_input.text in
the form of a string.
    datetime.datetime.now().strftime()
This function is called to return the current time
in the form of a string. We use this to write the
current time to a text file if an error occurs.
This only executes if an error occurs.
    os.getlogin()
This function is called to return the users
system username. We use this function to create
and manipulate text files. Think about it this way.
On linux /home/user/folder is a filepath.
The user portion of that refers to the users
logged in username. Without the current users
system username we can't write text files because we
don't know the full path to any safe locations for us
to write this data.
    sys.platform.startswith('platform')
This function is called to check the user computers
operating system. It checks a specific version number
for each style of operating system. For instance,
on windows this function checks registry keys and
on linux it makes use of system call to return the
major version string. On older linux systems this could
return linux2, or linux3, or linux4, or linux1 etc.. In
order to get around that we simply supply linux and parse
the string to dertmine the version number. On windows
it can return a variety of things such as win32.
Supplying win as the parameter guarantees that we
will determine if this is a windows based operating.
We use this so that we can write the files and
manipulate the program its self in a way that's
compatible with the various operating systems.
    self.ids.user_input.text
This function returns the string currently contained
in the user_input TextInput widget. We use this
to return the users string to the chatbot so that it
can formulate an appropriate response for the user.
As well as returning it to the appending of
self.master_log, self.__append_file(dat, path) etc.
    self.ids.user_input.focus
This function sets the current focus of the
```

```
users mouse to user_input TextInput widget.
Meaning that it's actively focused so the
user doesn't have to click back into it.
Unfortunately this is not having the desired
effect on windows operating systems due to an
ongoing issue with kivy and the windowing system
on windows.
    self.caprica_speak(words)
We call this function when the user has activated
either the audio or voice options. We pass the
chatbots generated response to this function.
We then access the systems text to speech software
to verbally "speak" the chatbots response to the
user.
    self.ids.view_port.text
We call this function to set the view_port TextInput
widgets text property. We set this property to contain
the users statement and the chatbots response in order.
User: This is a statement.
Caprica: Yes, that is a statement.
    self.record.listen(microphone_source)
We call this function to open the users microphone
assuming the user has a microphone. If they don'take
have a microphone the user wont be able to access the
voice option. If they do have a microphone this function
turns the microphone on and enables it to accept noise.
The noise is the users input, Ie, the users words. The
microphone remains in an active state as long as the
user speaks. That data is then piped into the recognizer
for transciption into a string.
    self.recognize_sphinx(audio_file)
This function is called after the user has made use
of the voice option. It takes the audio file piped
from the source variable to the audio variable which
stores this data as an audio file. This audio file is
then passed to self.recognize_sphinx(audio) which is
then transcribed to a string and returned.
Private Members
---------------
    self.__append_file(data, path)
We call this function to write specific data to a
specific text file. The text written to th files comes
in two flavors. All of the chatbots response are written
to the Caprica_Statements.txt file. All of the users
statements are written to User_Statements.txt file.
We use this to segregate the statements made by the user
and the chatbot for later training purposes.
    self._stop_threading()
This function is called to check to see if a thread is
active. If a thread is active this function interupts
the active thread which essentially (though not
technically) kills it; to prevent memory leaks and
a series of other potential issues.
Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
```

```
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error
log text file.
Returns
-------
    None
Notes
-----
    So this is a large function and I explained it quite
    well broken down in the sections above. An overview of
    this function is this.

    We check to see if the user has enabled or disabled the
    audio and voice options. We then accept the users input
    in a way appropriate to the option the user has elected
    to use. We then obtain a response from the chatbot and
    either send it to the user in text or audio form.
```

Definition at line 4725 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.20 get_caprica_text_response()

```
def Sentience.SentienceScreen.get_caprica_text_response (
              self )


def get_caprica_text_response(self)

Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

Attributes
----------
    self.username
Contains the username that user entered if he/she entered
one. The length of self.username is checked to see if it's
less than or equal to zero. If it is it means that the user
never set a username and a default value of "User: " is set
This value along with the users response to the chat bot and
'Caprica: ' and then the chat bots response are displayed
in text in self.ids.view_port TextInput Widget.


    temp
temp is used to store the response from the chat bot
temporarily.

    self.master_log
The string 'Caprica: ' and the value contained in temp are
added to the end of the string with a new line
character.

    self.ids.view_port
This is the main view port TextInput Widget here we briefly
store the User text response and chat box text response.

    self.ids.user_input
```

```
The string contained in the user_input TextInput Widget
is cleared and the hint_text reset.


Members
-------
    self.chatbot.get_response(self.user_input)
Obtains the response from the chat bot which is generated
to best fit the user response which is passed into this
function as a parameter. It then returns the chat bots
response and is in this case stored in the temp variable.
This is all type casted to str() to ensure type safety.

Private Members
---------------
    self.__append_file(self, words, path)
Is called to append the Chat bot responses to the text file
Caprica_Statements.txt


Returns
-------
    return None


Exceptions
----------
OSError
    The OSError can occur due to numerous reasons.
    What I'm primarily concerned with here however
    is import statements, incompatible Operating
    systems, and bad system calls. The exception
    if it occurs is handled and logged in an error
    log text file.

IOError
    The IOError can occur due to many reasons.
    My primary concern is file manipulation. The
    improper opening/closing/writing to files. If
    the exception occurs it's handled and logged; in
    an error log text file.

RunTimeError
    The RunTimeError error here is checking to make sure that
    the chat bot doesn't die. Essentially I just need to make
    sure that it completes and executes the python text to speech
    functions in a manner that doesn't cause a fatal exception. If
    something does occur the exception will be handled and logged to
    an error log text file.

ValueError
    Ensures that values passed to the chat bot are appropriate.
    And if for some reason one isn't the exception will be handled
    and logged to an error log text file.


Notes
-----
    **This function is deprecated and has been replaced
    with get_caprica_response**

    This function is called when self.audio_disabled == True

    It first checks the operating system.
    If sys.platform.startswith('linux') == True it executes
    the if statement intended for the linux operating system.

    Otherwise if sys.platform == False it executes the if
    statements intended for the windows operating system.

    It then enters the appropriate if statement
    and the user response that's stored in
    self.user_input is passed into
```

```
    self.chatbot.get_response(self.user_input) the generated
    response is then stored in the temp string variable.

    We then call self.__append_file('') we give it a new line
    character and the data stored in the temp variable and pass
    to the path parameter the apropriate path which is based off
    of the users operating system.

    We then set the string of the view_port TextInput Widget to be
    'Username: ' + user_response
    'Caprica: ' + caprica_response

    We then clear the seld.ids.user_input.text field
    (user_input TextInput Widget). So that the
    hint_text property is reset.
```

Definition at line 1231 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.21 get_caprica_voice_response()

```
def Sentience.SentienceScreen.get_caprica_voice_response (
              self,
              words )


    def get_caprica_voice_response(self, words)

    Parameters:
    -----------
param1 : self
    Denotes it as being a member of
    SentienceScreen(Screen) class.

param2 : words
    String variable containing the users
    response which will be used by the
    chat bot to generate an appropriate
    response to the users comment.

    Attributes
    ----------
temp
    The response generated by the chat bot is returned
    to the temp variable where it's stored as a string
    for later manipulation.

self.master_log
    A new line character plus the string 'Caprica: ' and
    the chat bots response are appended to the end of
    the self.master_log string variable.

    Members
    -------
self.chatbot.get_response(words)
    This function is called from a variety of
    locations. The string value passed to
    self.chatbot.get_response(words) is the users
    response to the chat bot. It's used to locate, and
    generate the best possible response from the chat bot.
    That response is then returned and stored in the
    variable temp.

self.caprica_speak(words)
    This function is called from a variety of locations.
    In this case it occurs when self.audio_enabled == True
    Once the function self.get_caprica_voice_response()
    has been called the users response gets sent to
```

```
    self.chatbot.get_response(words) which then causes
    the chat bot to come up with an appropriate response.
    Which is then returned to temp, temp is then passed
    to self.caprica_speak(temp) the string contained in
    the temp variable is then read by the systems speech
    to text software.

    Private Members
    ---------------
self.__append_file(string, path)
    This function is called to append the chat bots
    voice response to the Caprica_Statements.txt file
    along with a new line character.

    Returns
    -------
return None

    Exceptions
    ----------
OSError
    The OSError can occur due to numerous reasons.
    What I'm primarily concerned with here however
    is import statements, incompatible Operating
    systems, and bad system calls. The exception
    if it occurs is handled and logged in an error
    log text file.

IOError
    The IOError can occur due to many reasons.
    My primary concern is file manipulation. The
    improper opening/closing/writing to files. If
    the exception occurs it's handled and logged; in
    an error log text file.

RunTimeError
    The RunTimeError error here is checking to make sure
    that the chat bot doesn't die. Essentially I just
    need to make sure that it completes and executes the
    python text to speech functions in a manner that
    doesn't cause a fatal exception. If something does
    occur the exception will be handled and logged to
    an error log text file.

ValueError
    Ensures that values passed to the chat bot are
    appropriate. And if for some reason one isn't the
    exception will be handled and logged to an error
    log text file.

    Notes
    -----
**This function is deprecated and has been replaced
 with get_caprica_response**

When this function is called we first check to see
what operating system the user is running. If
system.startswith('linux') == True the user is
using a Linux based operating system. The appropriate
if statement is then executed.

Otherwise if sys.platform.startswith('win') == True
the user is using a windows based operating system. The
appropriate if statement is then executed.

We then call the function self.chatbot.get_response(words)
which is type casted to a string variable for safety. The
result of this function returns a generated response from
the chat bot and stores in the variable temp.

We then call the function self.__append_file(temp, path)
which appends a new line character and the contents of
the temp variable to the Caprica_Statements.txt file.
```

```
We next append the string 'Caprica: ' along with a new
line character and the contents of the temp variable to
the end of the self.master_log string variable.

Finally we call self.caprica_speak(temp) and pass the
temp variable to it. So that the text to speech software
can speak the generated response from the chat bot
contained in the temp variable.
```

Definition at line 1642 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.22 get_caprica_voice_thread()

```
def Sentience.SentienceScreen.get_caprica_voice_thread (
              self,
              words )


get_caprica_voice_thread(self, words)

Parameters
----------
    param1 : self
Denotes this as being a member of SentienceScreen().
    param2 : words
A string containing the users transcribed voice response
is passed to this for later manipulation.
Attributes
----------
    temp
temp is a string variable that is used to temporarily
store the generated response of the chatbot. This
variable will then be written to varios files and
displayed in the view_port TextInput widget.
    self.master_log
self.master_log is a string variable wich contains a
master conversation log. This log includes the text
sent by the user and the responses generated by
the chatbot as they appear.
    self.username
self.username is a string variable which contains the
users chosen username. IF the user did not elect to
setup a user profile a default value of 'User: ' is
provided.
Members
-------
    self.caprica_speak(words)
This function is called after the user has sent
text to the chatbot. That text is then passed to
this funtction if self.audio_enabled == True
or if self.voice_enabled == True. We then
access the users systems text to speech software
to verbally speak the passed string. This function
is a member of SentienceScreen().
    time.sleep(integer)
We call time.sleep(1) to force the program to
sleep for one second. This ensures that certain
functions are called by forcing the Kivy.clock() to
appropriately execute events in the correct order
in the frame. It also prevents the program from hanging
by trying to execute things to fast. This function is a
member of the class time.
    sys.platform.startswith(string)
We call this function to determine the users operating
system. If the user is running a windows system then
the appropriate if statements execute. If they're
```

```
running a linux system again the appropriate if
statements are executed. We determine this by accessing
the systems major version. For instance, older
linux systems return values such as, linux1, linux2,
linux3 and so on. Windows systems may return win32 etc.
By checking the preceeding version string, Ie, 'linux'
we know it's a linux system, or 'win' we know it's a
windows system. Where these strings comes from varies
based on the operating system. On linux it's an os call.
On windows it's a registry key. This function is a
member of the class sys.
    datetime.datetime.now().strftime(string)
We call this function to return the current local time.
We format it to ourput in
year, month, day, hours, minutes seconds. This is
returned as a string directly to our write method.
    os.getlogin()
This function is a member of the os class. We call
this function to return the current users system user
name. We do this so that we can succesfully write files
to the users system. On linux the filesystem has the
users name as part of it's non root path. We need this
name to access the location where we want to write to.
Private Members
---------------
    self.__append_file(string, path)
This function is a member of SentienceScreen(). We
call this function to append specific data to
specific text files. The data is passed in as a string.
As is the path to the file.
    self._stop_threading()
This function is a member of the SentienceScreen()
class. we call this function to interupt our running
threads.
    self.__currently_thinking(boolean)
This function is a member of the SentienceScreen()
class. We call this function to change the current
banner which informs the user if the program is
currently inactive or thinking. It change the text to
either '...Thinking...' with a red foreground. Or,
'...Inactive...' with a blue foreground. The boolean
variable dictates whether or not the program is in fact
actively thinking or not.
Exceptions
----------
     OSError
 The OSError can occur due to numerous reasons.
 What I'm primarily concerned with here however
 is import statements, incompatible Operating
 systems, and bad system calls. The exception
 if it occurs is handled and logged in an error
 log text file.

     IOError
 The IOError can occur due to many reasons.
 My primary concern is file manipulation. The
 improper opening/closing/writing to files. If
 the exception occurs it's handled and logged; in
 an error log text file.

     RunTimeError
 The RunTimeError error here is checking to make sure
 that the chat bot doesn't die. Essentially I just need
 to make sure that it completes and executes the python
 text to speech functions in a manner that doesn't cause
 a fatal exception. If something does occur the
 exception will be handled and logged to an error log
 text file.

     ValueError
 Ensures that values passed to the chat bot are
 appropriate. And if for some reason one isn't the
 exception will be handled and logged to an error
```

```
 log text file.
Returns
-------
    None
Notes
-----
    This function is one our response threads. We call it
    if the user has activated the voice feature. Ie, if
    self.voice_enabled == True, this function will be called
    when the user clicks on the enable/disable voice button
    which is represented by a red or blue microphone on the
    menu bar (Action Bar).

    We determine the users operating system. Obtain the chatbots
    generated response. Next we append it to
    the Caprica_Statements text file. We then save it to the
    master log. We then display the response and the users
    initial text in the view_port TextInput widget. Next we
    call self.caprica_speak(words) to actually verbally
    communicate the chat bots response to the user.

    We next interupt the thread and put the program to sleep
    for one second by calling time.sleep(1). We finally
    call self.__currently_thinking(False) to reset the
    banner text to '...Thinking...' with a blue foreground.
```

Definition at line 5143 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.23 get_user_text()

```
def Sentience.SentienceScreen.get_user_text (
              self )


We call this function to return the text contained in
self.ids.user_input.text TextInput Widget; in the form of a
string.
```

Definition at line 5667 of file Sentience.py.

Here is the caller graph for this function:

6.5.3.24 get_user_text_response()

```
def Sentience.SentienceScreen.get_user_text_response (
              self )


def get_user_text_response(self)

Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

Attributes
----------
    self.audio_disabled
Preforms a check to establish wether or not
self.audio_disabled == True.

    self.user_input
```

```
       Accepts text (string) from user_input TextInput Widget.

    self.ids.user_input
TextInput Widget that returns its current string to
self.user_input. Or self.get_caprica_voice_response()

    self.master_log
Stores the contents of self.ids.user_input TextInput
string along with some other data.

    self.audio_enabled
Checks to see if self.audio_enabled == True

Members
-------
    self.get_caprica_voice_response(words)
Is called when self.audio_enabled == True Accepts
self.ids.user_input.text as its parameter. That is to
say that self.ids.user_input returns its string to this
function. The chat bot then searches its database and
locates the best  possible response. It then calls
self.caprica_speak() to read that response.

Private Members
---------------
    self.__append_file(path)
This is called to append the user_input value to the
User_Statements.txt file.

Returns
-------
    return self.get_caprica_text_response()
Ends with the text response of caprica_speak() which is
a resposne to the users comment.

    return None
Function returns None when self.audio_enabled == True

Exceptions
----------
OSError
    The OSError can occur due to numerous reasons.
    What I'm primarily concerned with here however
    is import statements, incompatible Operating
    systems, and bad system calls. The exception
    if it occurs is handled and logged in an error
    log text file.

IOError
    The IOError can occur due to many reasons.
    My primary concern is file manipulation. The
    improper opening/closing/writing to files. If
    the exception occurs it's handled and logged; in
    an error log text file.

FileNotFoundError
    FileNotFoundError is exactly what it sounds like.
    If the file I'm trying to write to doesn't exist
    we may have a problem. But not to worry it's handled
    and logged.


Notes
-----
    **This function is deprecated and has been replaced
    with get_caprica_response**

    This function is called when the user hits the "Enter" key
    on their key board while clicked into the
    self.ids.user_input TextInput Widget.

    What happens next is determined by the value(s) of
    self.audio_enabled and self.audio_disabled.
```

```
    if self.audio_enabled == True the chat bot obtains the users
    comment and then calls self.caprica_speak() where it
    essentially returns None

    if self.audio_disabled == True the chat bot obtains the users
    comment passes that string to the
    self.get_caprica_text_response(string) and then returns
    self.get_caprica_text_response(string)
```

Definition at line 1069 of file Sentience.py.

Here is the call graph for this function:

6.5.3.25 get_user_voice_response()

```
def Sentience.SentienceScreen.get_user_voice_response (
              self )


def get_user_voice_response(self)

Parameters:
-----------
    param1 : self
Denotes it as being a member of
SentienceScreen(Screen) class.

Attributes
----------
    self.voice_disabled
Checks to see if self.voice_disabled == True or
False If self.voice_disabled == True a warning
message is sent to the user in forming them that
they need to click on the red microphone image;
before clicking the "Record user" button. The
warning message is sent is displayed in
self.ids.view_port. If self.voice_disabled == False
self.voice_enabled == True and the user can begin
recording their voice via a microphone.

    self.ids.view_port
A warning message is displayed in the view_port
TextInput Widget informing the user they need to
first enable the voice option before they can use
their microphone to speak with the chat bot.

    self.mic
self.mic = sr.Microphone() this is what enables us to use
the microphone to speak with the chat bot.

    source
When we begin recording the users voice we do so in a
loop we open that loop (open the microphone) as source.
All the audio detected is piped into source and then
stored in the audio variable.

    audio
The recorded sound will be streamed to the function
self.record.listen(source) all audio picked up will
be saved in the audio variable for later transcription
into a string.

    temp
The data contained in the audio variable is passed to
the function self.record.recognize_sphinx(source) which
will then transcribe the audio file and store the
returned string in the temp variable.
```

```
    self.master_log
A new line character, the users username and the string
stored in the temp variable are then appended to the
self.master_log string.

Members
-------
    self.record.listen(source)
This function which is derived from
speech_recognition.Recognizer()is called when we open
the loop for recording the users voice response. Source
is the stream for the microphone. The data contained in
source is piped into this function as its parameter and
then returned to the audio variable.

    self.record.recognize_sphinx(audio)
The data contained in the audio variable is sent to this
function for transcription into a string. The data once
transcribed is returned to the temp variable for later
manipulation.

    self.get_caprica_voice_response(string)
This function is called after the users voice response
has been transcribed and stored in the temp variable.
The temp variable is then passed into this function as
its parameter which then obtains the most accurate result
possible for the chat bot to respond to the user. This
response is then spoken by the self.caprica_speak()
function.

Private Members
---------------
    self.__append_file(string, path)
This function is called to append the users transcribed
voice response to the User_Statements.txt file along with
a new line character.

Returns
-------
    return None


Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the exception
will be handled and logged to an error log text file.

    speech_recognition.Recognizer().UnknownValueError
```

```
This exception can occur in a variety of ways but the
primary concern for me. IS when the Recognizer() is
unable to interpret the users voice response. If this
exception occurs it's handled logged to an error logs
text file.

    speech_recognition.Recognizer().RequestError
This exception can occur for a variety of reasons but
the primary concern is when we're unable to open the
microphone. That is to say when no microphone is detected.
If it occurs the exception is handled and logged in an
error logs text file.


Notes
-----
    **This function is deprecated and has been replaced
    with get_caprica_response**

    When this function is called we first check to see if the
    user is running a Linux or windows based operating system.

    If sys.platform.startswith('linux') == True then the user
    is using a Linux Operating system and the appropriate if
    statements will execute.

    Otherwise if sys.platform.startswith('win') == True
    Then the user is using a windows based operating system
    and the appropriate if statements will execute.

    We next to see if self.voice_disabled == True if it is
    True we issue a warning to the user telling them to first
    click on the "Enable/Disable Microphone" button which
    is represented by a red microphone when disabled or a blue
    microphone when enabled. This warning message is sent to the
    view_port TextInput Widget.

    We then open the microphone for listening and when audio is
    detected it's piped into the recognizers listening function.
    When no more audio is detected its stored in the audio
    variable. The loop then ends.

    We then call the function to transcribe the audio into a
    string that data is then returned to the temp variable.

    We then append the contents of the temp variable and a
    new line character to the User_Statements.txt file.

    Next we append a new line character, the users Username,
    and the contents of temp to the self.master_log string.

    Finally, we pass the temp variable to
    self.get_caprica_voice_response(str(temp)) which then
    calls the self.caprica_speak() function to speak the
    generated response to the user.
```

Definition at line 1400 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.26 increase_chatbot_volume()

```
def Sentience.SentienceScreen.increase_chatbot_volume (
              self,
              vol )


    Increase the chatbots volume by vol when called.
    Chatbots volume can only be set between 0-1
```

```
    with 0 being the lowest and 1 being the highest volume
    level.

    If vol > 1 or < 0 a warning message is displayed in the
    view_port TextInput widget.
```

Definition at line 1025 of file Sentience.py.

6.5.3.27 increase_rate_of_speech()

```
def Sentience.SentienceScreen.increase_rate_of_speech (
              self,
              value )
```

```
    Increases the words per minute spoken by value.
    Increasing this value increase teh rate of speech
    and may make the chatbots verbal messages incoherent.
```

Definition at line 1211 of file Sentience.py.

6.5.3.28 on_mouse_pos()

```
def Sentience.SentienceScreen.on_mouse_pos (
              self,
              instance,
              pos )
```

```
def on_mouse_pos(self, instance, pos):
    This function is called everytime that the user moves
    the mouse. It checks to see if the mouse is colliding
    (hitting) any of the widgets on the menu bar. In this
    case, I focus on the buttons. If the mouse touches
    any of the buttons a tooltip is created and displayed
    where the mouse was located; explaining what that
    particular button does.


Parameters:
-----------
    param 1: self
Denotes this as being a member of SentienceScreen()

    param 2: instance
Returns the current "instance" of the mouse. Similar
to coordinates in that it refers to "This current
position". If the mouse moves again its instance has
changed.

    param 3: pos
The current coordinates of the mouse as it relates
to the window.


Attributes
----------
    colliding_computer = self.ids.select_os.collide_point(*pos)
colliding_computer stores the collision point (the
coordinates of the "Select OS" button).
```

```
    colliding_record = self.ids.record_user.collide_point(*pos)
 colliding_record stores the collision point (the
 coordinates of the "Record user" button).

    colliding_voice = self.ids.voice_enable_disable.collide_point(*pos)
 colliding_voice stores the collision point (the
 coordinates of the "Enable/disable voice" button).

    colliding_audio = self.ids.audio_enable_disable.collide_point(*pos)
 colliding_audio stores the collision point (the
 coordinates of the "Enable/Disable audio" button).

    colliding_eraser = self.ids.erase_text_button.collide_point(*pos)
 colliding_eraser stores the collision point (the
 coordinates of the "Erase text" button).

    colliding_pencil = self.ids.write_file_button.collide_point(*pos)
 colliding_computer stores the collision point (the
 coordinates of the "Write Logs" button).

    colliding_printer = self.ids.print_logs.collide_point(*pos)
 colliding_computer stores the collision point (the
 coordinates of the "Print logs" button).

    self.ids.select_os
This is a reference to the select_os Button widget.

    self.ids.record_user
This is a reference to the record_user Button widget.

    self.ids.voice_enable_disable
This is a reference to the voice_enable_disable
Button widget.

    self.ids.audio_enable_disable
This is a reference to the audio_enable_disable
Button widget.

    self.ids.erase_text_button
This is a reference to the erase_text_button
Button widget.

    self.ids.write_file_button
This is a reference to the write_file_button
Button widget.

    self.ids.print_logs
This is a reference to the print_logs Button widget.

    self.tooltip_open
This is a member of SentienceScreen(). This is how
we determine if a tooltip is currently open. If this
is open we then know we need to close it and set it
to self.tooltip_open = False

    self.tooltip.pos
This is a member of the ToolTipLabel widget. We simply
set (or reset by setting it) the current position of this
widget to the position of the instance of the pointer
which collided with this calling function. I.e,.
If the mouse collides wit the select_os button
then we use that exact collision point to set
the position of this widget and then add the tooltip
at that position.


Members
-------
    self.ids.select_os.collide_point(*pos)
Is called when the current instance and position of the
mouse collide (touch/hit) the select_os button widget.
```

```
    self.ids.record_user.collide_point(*pos)
 Is called when the current instance and position of the
 mouse collide (touch/hit) the record_user button widget.

    self.ids.voice_enable_disable.collide_point(*pos)
 Is called when the current instance and position of the
 mouse collide (touch/hit) the voice_enable_disable
 button widget.

    self.ids.audio_enable_disable.collide_point(*pos)
 Is called when the current instance and position of the
 mouse collide (touch/hit) the audio_enable_disable
 button widget.

    self.ids.erase_text_button.collide_point(*pos)
 Is called when the current instance and position of the
 mouse collide (touch/hit) the erase_text_button button
 widget.

    self.ids.write_file_button.collide_point(*pos)
 Is called when the current instance and position of the
 mouse collide (touch/hit) the write_file_button button
 widget.

    self.ids.print_logs.collide_point(*pos)
 Is called when the current instance and position of the
 mouse collide (touch/hit) the print_logs button widget.

    self.get_root_window()
This function applies to the root window. It's called
as a check when the users access the tooltips. The
check preformed ensures that if the users moves the
mouse out of the programs window the tooltip widget
is destroyed.

    self.set_tooltip_text(text)
We call this function to set the tooltip text.
We do this each time a tooltip is created but we only
change the text based on the widget that the mouse
collided with. We don't want the user to see "Select Os"
when they collide with the print_logs button
when they should see "Print file".

    self.display_tooltip(*args)
We finally call this function actually
add a new label widget, which is our tooltip,
to the screen.

    # Todo: update documentation to reflect current status.
Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
   None

Notes
-----
    this function is called whenever the user moves his or her
    mouse. It only ever "does something" when the mouse collides
    with a widget listed in the conditional statements. In this
    case, when the users mouse touches (collides) with one of
    the buttons on the menu bar. When that happens the if
    statements are checked and we determine which widget
    the mouse has collided with.

    Once we've determined what widget the users mouse has
    collided with. We then set self.tooltip_open = True
```

```
    We set the position of the ToolTipLabel to the
    position of the users mouse when that mouse collided
    with that specific widget.

    We then specify what text we want the tooltip to
    display as it relates to that specific widget.

    Finally we call the function to create and add that
    widget to the screen.
```

Definition at line 4006 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.29 onEnd()

```
def Sentience.SentienceScreen.onEnd (
              self,
              name,
              completed )


def onEnd(self, name, completed)
    We call this function when self.caprica_speak(string)
    ends. To be more precise it's called every time that
    self.engine.say(string) is finished. This function
    kills the event loop and empties the event queue.

Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

    param2 : name
The parameter name is in reference to the name
of the event that self.onEnd() is bound to. In
this case the event is 'finished-utterance'.

    param3 : completed
The parameter completed is in reference to the
function. In this case the calling function. Which
is self.engine.say(string).


Attributes
----------
    None

Members
-------
    self.engine.endLoop()
This is function is called when self.caprica_speak()
finishes speaking the string passed to it. The purpose
of this function is to empty the event queue and Ensures
that all strings have been processed in said queue.
It relates to self.engine.say(string).

Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
    OSError
```

```
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure that
the chat bot doesn't die. Essentially I just need to make
sure that it completes and executes the python text to
speech functions in a manner that doesn't cause a fatal
exception. If something does occur the exception will be
handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are appropriate.
And if for some reason one isn't the exception will be
handled and logged to an error log text file.

    Notes
    -----
The purpose of this function is simply to terminate the
self.caprica_speak() function. Aside from the Operating
system check and the exceptions; there is only one line
which is the terminating function for the
self.engine.startLoop() function.
```

Definition at line 2229 of file Sentience.py.

Here is the call graph for this function:

6.5.3.30 open_delete_file_dialog()

```
def Sentience.SentienceScreen.open_delete_file_dialog (
              self )


open_delete_file_dialog(self)

Parameters
----------
    param1 : self
Denots this as being a member of SentienceScreen().
Attributes
----------
    content
content is local variable to the function
self.open_delete_file_dialog() we use this
to make a new Object. This object is DeleteDialog
we then add the instantiated object to Popup().
Note: We add the local object content which is
the DeleteDialog to the kivy Popup() content field.
We do this because of how the layouts work.
    delete_file
delete_file is the ObjectProperty() that we
created in the DeleteDialog() class. We're using
this property, to bind it to the
SentienceScreen.().delete_file() function.
    self.popup
self._popup is the decleration and initialization
of the kivy Popup(). We've created this object, this
```

```
Popup() window and can now call it at anytime.
    Cancel
Cancel is the ObjectProperty() that we
created in the DeleteDialog() class. We're using
this property, to bind it to the
SentienceScreen.().dismiss_popup() function.
    title
title is a member of the Kivy Popup() class. It's
a StringProperty() which we use to set the title of
the Popup() window.
    size_hint
size_hint is member of the Kivy Popup() class, and
all other kivy widgets. We use size_hint to to set
the size of the widget. This enables us to set a
size based of percentages of the users monitor size.
Simply put, this enables us to create a size that's
compatiable with all devices and will stretch and
shrink in a manner that wont distort the programs
appearance.
Members
-------
    Popup()
Popup() is a kivy widget which is exactly what it
sounds like. It's a popup window. It's not a new window,
it's a widget wich locks to the MainWindow (root window);
we use this to allow the user to navigate to a
specific file, select that file, and then delete it.
    DeleteDialog(FloatLayout)
DeleteDialog is a class with a default FloatLayout
that we created earlier. We use this class and its
layout with our Popup() widget.
    self._popup.open()
We call this function to add the Popup() widget
to the screen.
    self.delete_file(self, path, filename)
self.delete_file(self, path, filename) is the
same function that we bound to the delete_file
ObjectProperty; we just ommited its parameters
when we did it. It recieves its arguements when
the users selects a file (clicks on one) in the
Popup() window and then clicks the delete button
in the Popup() window. This function is passed
the name of the file and it's file path. I
actually don't use all of the parameters. But
both of them could be used.
    self.dismiss_popup()
We call this function to delete the Popup() window
from the screen.
       Private members
       ---------------
   None
Returns
-------
    None
Exceptions
----------
    None
Notes
-----
    This function is pretty straight forward. When the user
    clicks on the "Delete File" button a popup window is
    created and then added to the screen. This window contains
    a file browser, and two buttons. The file browser allows
    the user to navigate through their file system and select
    a file that they wish to delete.

    Once the user has located the file they wish to delete
    they simply click on that file, which selects it, and
    then click the "Delete" button in the popup window.

    This then returns the filename and path of the file to
    the self.delete_file(self, path, file). Which then preforms
    the deletion operation.
```

```
    We then finally close the popup by removing (deleting) it
    from the MainWindow.
```

Definition at line 5677 of file Sentience.py.

Here is the call graph for this function:

6.5.3.31 open_print_file_dialog()

```
def Sentience.SentienceScreen.open_print_file_dialog (
              self )


def open_print_file_dialog(self): calls self.open_print_file_dialog
when the user clicks on the "Print" button on the menu bar.

content = PrintDialog(print_files = self.print_files,
Cancel = self.dismiss_popup) sets the ObjectProperty(s)
of PrintDialog class to reference the local and or instance
variables, functions (in this case)

self._popup = Popup(title = "Print File", content = content,
size_hint = (0.9, 0.9)) create Popup, set title to 'Print File',
content = content (PrintDialog), size_hint 90% width,height

self._popup.open() : calls function to open the popup
```

Definition at line 3972 of file Sentience.py.

Here is the call graph for this function:

6.5.3.32 print_files()

```
def Sentience.SentienceScreen.print_files (
              self,
              path,
              filename )


def print_files(self, path, filename)
    This function is called by the function
    self.open_print_dialog(self) function which is
    called by clicking on the 'Print' button on the
    menu bar. A new Popup() window is created
    which allows the user the ability to navigate to
    and select a specific file which they want to print.
    Once the user has selected that file they can click they
    'Print' button on the bottom bar of the Popup() window.
    Which then calls this function.


Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

    param2 : path
This is the path to the file. Note:
including the file name is redundant. If the
selection tool for the 'Select file' Popup()
window function is re-written. It can return the
full path and not separate it.
```

```
    param3 : filename
The name of the file to be printed. Note:
This is redundant see the param2 explanation.

Attributes
----------
    temp
The temp variable is a string variable.
This variable joins the path and filename
parameters to gain the absolute path of the
file to be printed.
temp = str(path) + str(filename) they're type
casted for safety.

    path
The path variable stores the path to the
 file that user has selected and wishes to print.
 This is passed along with file name when the
 user clicks the 'print' button on the bottom
 bar of the PrintDialog() Popup() window.

    filename
The filename variable stores the file name of the
file that user has selected and wishes to print.
This is passed along with the path when the
user clicks the 'print' button on the bottom
bar of the PrintDialog() Popup() window.

    toBytes
toBytes is exactly what it sounds like. When Linux
users access this print_files(self, path, filename)
function. The path and file name are created as a single
string. Which is then converted to a bytes object for
printing. Note: This is redundant, you'll note in the
windows section of the code that I've simply used the
built in cast for the string class to encode the string
as it's passed to the native print function. That is to
say str.encode('') which returns the encoded string. I
could and should do that for the linux section as well.

    import win32api
If the user is using a windows based operating
system. This is imported in that section. This
import gives us access to the native windows32
api print calls. Note: This is not being used
right now as I'm testing a new way of doing this
that is more pythonic than calling the windows Shell
directly. If this import statement exists outside of
this function the program will not run. Because it
will cause a fatal import error on Linux systems.

    import win32print
If the user is using a windows based operating
system. This is imported in that section. This
import gives us access to the native windows32
api print calls. Note: This is not being used
right now as I'm testing a new way of doing this
that is more pythonic than calling the windows Shell
directly. If this import statement exists outside of
this function the program will not run. Because it
will cause a fatal import error on Linux systems.

    lpr
This directly access the printer driver on a Linux based
operating system.

    stdin
while creating the lpr object we set the stdin variable
to access the subprocess call to subprocess.PIPE. From
this call we're able to open and read in a file that's
contents will be piped to the variable for printing.
```

```
Members
-------
    win32api.ShellExecute()
Executes a windows shell to directly call
the windows 32 api printer calls. Uses
win32print.GetDefaultPrinter() to return
and select the active printer.

    win32print.GetDefaultPrinter()
This function is exactly what it sounds like.
It returns the default system printer and
when accessed and called as it is in this
function the default printer ID is returned
in the position of 'what printer do I send
this file to'.

    lpr.stdin.write()
Takes the data stored in the variable toBytes pipes it
to the active printer.

    subprocess.Popen()
Opens the active printer by directly accessing the
driver and then the default system printer.

    subprocess.PIPE
Allows us to pipe the data in toBytes to the active
printer.

    os.startfile('')
This is a pyhtonic command which opens the file and
then if told to via 'print' string, sends the
specific file to the default printer.

Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error log
text file.

    FileNotFoundError
```

```
This can occur in a variety of ways however my primary
concern is that file path the user selected is broken.
Resulting in an File Not Found error. If this occurs
it's handled and logged to an error file text log.

    NameError
Again this can occur in a variety of ways but the
primary concern is that the conversion to bytes does
not take place or breaks some how due to wacky Unicode
characters. In which case the exception is handled and
logged to an error log text file.
Notes
-----
    When this function is called we check the users operating
    system. If sys.platform.startswith('linux') == True we know
    that the user is using a linux based operating system. In
    which case the appropriate if statements are executed.

    Otherwise if sys.platform.startswith('win') == True then we
    know that the user is using a windows based operating system.
    In which case the appropriate if statements are executed.

    For linux users the process is relatively straight forward.
    We directly access the printer driver in user/bin we
    determine the active printer. We then write the stream to
    said printer to actually print the file.

    For windows users we have two methods though one is
    commented out. The active method is the pythonic version.
    os.startfile() We pass the file path and the 'print' string
    to let the function know that we mean to print the file at
    location path. It does the same thing the commented out
    section does it just cuts out those steps and uses pythons
    built in os library. Which preforms those steps behind the
    scenes.
```

Definition at line 3045 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.33 set_enable_disable_audio()

```
def Sentience.SentienceScreen.set_enable_disable_audio (
              self )


def set_enable_disable_audio(self)
    This function is called when the user clicks on the
    enable/disable audio button; which is represented by
   the red or blue speaker button.

Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

Attributes
----------
    self.audio_disabled
If self.audio_disabled == True then the chat bots
audio function is disabled. This means that the chat
bot can only communicate with the user via text. If
self.audio_disabled == True it's represented by a
red speaker image on the menu bar. Clicking on the
red speaker image will activate the audio and turn
the red speaker image blue.

    self.audio_enabled
```

```
If self.audio_enabled == True then the chat bot can
access the systems text to speech software and
verbally read the string passed to
self.caprica_speak(self, words) back to the user. If
self.audio_enabled == True; it's represented by a blue
speaker image on the menu bar. Clicking on the blue
speaker will disable the audio and turn the image of
the speaker red.

    self.ids.user_input
If self.audio_enabled == True or
self.audio_disabled == True we set the
opacity of the user_input TextInput widget
to 1 making it visible. By default it's already
visible but if the user enables the microphone
option; all widgets not on the menu bar have their
opacity set to 0.

    self.ids.view_port
If self.audio_enabled == True or
self.audio_disabled == True we set the
opacity of the view_port TextInput widget
to 1 making it visible. By default it's already
visible but if the user enables the microphone
option; all widgets not on the menu bar have their
opacity set to 0.

    self.ids.audio_enable_disable
After the user has clicked on either the red or
blue speaker and the appropriate if statements
are executed based on the users operating system.
We change the icon property of
self.ids.audio_enable_disable and set the icon to the
appropriate image on the menu bar.


Members
-------
    self.caprica_speak(string)
We call this function to alert the user
to the current status of the audio function.
If the audio has been enabled the user is
informed that the audio is now active. If
the audio has been disabled the user is informed
that the audio feature has been disabled.

Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
```

```
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error log
text file.

Notes
-----
    This function is called when the user clicks
    either the blue or red speaker image on the menu bar.

    The users operating system is then checked. If
    sys.platform.startswith('linux') == True then
    the user is using a Linux based operating system.
    The appropriate if statement is then executed.

    Otherwise if sys.platform.startswith('win') == True
    then the user is using a windows based operating system
    and the appropriate if statements are executed.

    If self.audio_disabled == True
    the audio option is disabled and the speaker image is red.
    Clicking on the red speaker image will enable the audio
    feature.

    If self.audio_enabled == True the audio option is active
    clicking on the blue speaker image will disable the audio
    feature.
```

Definition at line 2559 of file Sentience.py.

Here is the call graph for this function:

6.5.3.34 set_enable_disable_voice()

```
def Sentience.SentienceScreen.set_enable_disable_voice (
              self )


def set_enable_disable_voice(self)
    This function is called when the user clicks on the
    enable/disable voice button; which is represented by
    the red or blue microphone button.

Parameters:
-----------
    param1 : self
Denotes it as being a member of SentienceScreen(Screen)
class.

Attributes
----------
    self.voice_disabled
If self.voice_disabled == True then the users voice
function is disabled. This means that the user can
only communicate with the chat bot via text. If
self.voice_disabled == True it's represented by a
red microphone image on the menu bar. Clicking on the
red microphone image will activate the voice function
and turn the red microphone image blue.

    self.voice_enabled
If self.voice_enabled == True then the user can
```

```
access their plugged in or on-board microphone to
verbally communicate with the chat bot if
self.voice_enabled == True; it's represented by a blue
microphone image on the menu bar. Clicking on the blue
microphone will disable the voice function and turn the
image of the microphone red.

    self.ids.user_input
If self.voice_enabled == True we set the
opacity of the user_input TextInput widget
to 0 making it invisible. By default it's
visible but if the user enables the microphone
option; all widgets not on the menu bar have their
opacity set to 0.

    self.ids.view_port
If self.voice_enabled == True we set the
opacity of the view_port TextInput widget
to 0 making it invisible. By default it's
visible but if the user enables the microphone
option; all widgets not on the menu bar have their
opacity set to 0.

    self.ids.voice_enable_disable
After the user has clicked on either the red or
blue microphone and the appropriate if statements
are executed based on the users operating system.
We change the icon property of
self.ids.voice_enable_disable and set the icon to the
appropriate image on the menu bar.

    self.mic
self.mic is our sr.Microphone() object
this is what enables us to accept the
users voice via microphone.

    source
If the user is using a Linux based operating
system we open their microphone as source.
The recorded audio is stored in source and
then passed to the self.adjust_for_ambient_noise()
function which sets the self.record.energy_threshold
value.

    self.ids.record_user
This is the "Record user " button. If
self.voice_enabled == True this button which
is located on the menu bar is represented by a blue
talking head. If self.voice_disabled == to True then
this button is represented by a red talking head. If
the user clicks on the head when it's blue they can
begin speaking into their microphone. The user should
speak as clearly as possible and then be silent for
10-20 seconds. If the user clicks on this button
when it's red a warning message will be given to
the user in the view_port TextInput Widget. Stating
that the user must first enable the voice feature.

    self.record.energy_threshold
Note: This applies to linux
We open the users microphone (activate) as source
we then listen to sounds being produced over a
specific threshold, so if sound is greater than
energy_threshold x accept audio as valid.
Basically we open the microphone and begin
recording the sound until the sound stops.
Note: This applies to windows
If the user is using a windows based operating
system. I've elected to set this value manually
to 1000 due to a higher sensitivity issues on
windows operating systems.
Members
-------
```

```
    self.caprica_speak(string)
We call this function to alert the user
to the current status of the voice function.
If the voice has been enabled the user is
informed that the voice feature is now active.
If the voice feature has been disabled the user
is informed that the voice feature has been disabled.

    self.record.adjust_for_ambient_noise(source)
self.voice_enable_disable is called and the
appropriate if statements are executed based
on the users operating system. if the users
operating system is Linux based. We open the
users microphone and record all audio until
no more audio is detected. We store this
in the variable source which is passed to
self.record.adjust_for_ambient_noise(source)
which sets the self.record.energy_threshold
value. This value attempts to compensate
for background noise in an attempt to make the
future audio transcription process more accurate.

Private Members
---------------
    None

Returns
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error log
text file.

    sr.UnknownValueError
This exception can occur in a variety of ways but the
primary concern for me. IS when the Recognizer() is
unable to interpret the users voice response. If this
exception occurs it's handled logged to an error logs
text file.

    sr.RequestError
This exception can occur for a variety of reasons but
the primary concern is when we're unable to open the
microphone. That is to say when no microphone is detected.
If it occurs the exception is handled and logged in an
```

```
error logs text file.

Notes
-----
    We first check the users operating system. If
    sys.platform.startswith('linux') == True
    then the user is using a linux based operating system
    and the appropriate if statements execute.

    Otherwise if sys.platform.startswith('win') == True
    the user is using a windows based operating system
    and the appropriate if statements execute.

    if self.voice_disabled == True then the voice mode
    is currently disabled. However, by clicking on the red
    microphone image the user has enabled the voice mode.

    We then set self.voice_enabled == True
    and self.voice_disabled == False
    We next set the opacity of
    self.ids.user_input = 0
    and self.ids.view_port.opacity = 0
    I chose to disable (hide) all widgets except for
    those on the menu bar when the voice mode is enabled.

    We then call self.caprica_speak() to inform the user
    that the voice mode has been enabled. We change the
    icon of self.voice_enable_disable to a blue microphone
    and the icon of self.record_user is set to a blue talking
    head.

    We next activate the users microphone and begin recording
    all sound until that sound stops. This sound is stored
    in the variable source which is passed to
    self.record.adjust_for_ambient_noise(source) which is
    used to set the value of self.engine.energy_threshold
    which is value we use to attempt to compensate for any
    background nose (interference). We only activate the
    microphone on linux systems. On windows systems the value
    of self.record.energy_threshold is set by default to 1000.
    I do this because their is a much higher sensitivity level on
    windows than on linux.

    If self.voice_enabled == True then clicking on the blue
    microphone image will disable the voice feature. This
    follows the same process as the enabling feature.

    We shut the microphone off change the images on the menu
    bar to their red counterparts, and show all the hidden
    widgets.
```

Definition at line 2746 of file Sentience.py.

Here is the call graph for this function:

6.5.3.35 set_gender()

```
def Sentience.SentienceScreen.set_gender (
              self )


    def set_gender(self)

    Parameters:
    -----------
param1 : self
    Denotes it as being a member of SentienceScreen(Screen) class.

    Attributes
```

```
    ----------
voices
    This will hold a list of all the systems text to speech
    voices. Ie, All of the voices installed in your tts software
    will be available, but I've chosen the voice for you. The list
    is sourced from the self.engine.getProperty('voice') call.



Members
-------
    self.engine.setProperty('voice', 'english+f2')
        This function is called in order to select and
        set a specific property of self.engine() which
        is the pyttsx3 tts library. In this call we're
        setting the voice property (Female, male, etc..)
        to female with the second parameter string. This
        handled in a slightly different manner between
        Linux and Windows, though the difference is
        minimal. It's different because the Windows voice
        file is the registry key and it's easier to
        manipulate voice properties as a list.

    self.engine.getProperty('voices')
        This call returns the list of available voices
        to the voices string.


    Private Members
    ---------------

    Returns
    -------
return None

    Exceptions
    ----------
OSError
    The OSError can occur due to numerous reasons.
    What I'm primarily concerned with here however
    is import statements, incompatible Operating
    systems, and bad system calls. The exception
    if it occurs is handled and logged in an error
    log text file.

IOError
    The IOError can occur due to many reasons.
    My primary concern is file manipulation. The
    improper opening/closing/writing to files. If
    the exception occurs it's handled and logged; in
    an error log text file.

RunTimeError
    The RunTimeError error here is checking to make
    sure that the chat bot doesn't die. Essentially
    I just need to make sure that it completes and
    executes the python text to speech functions in a
    manner that doesn't cause a fatal exception. If
    something does occur the exception will be handled
    and logged to an error log text file.

ValueError
    Ensures that values passed to the chat bot are
    appropriate. And if for some reason one isn't
    the exception will be handled and logged to an
    error log text file.

    Notes
    -----
This function is called in
SentienceScreen().__init__(self, **kwargs)  call. the
purpose of this function is to change the default text to
speech voice to female. The chat bot is named Caprica
```

```
and Caprica is a female.

When the function first runs we check to see what
operating system the user is running. If
sys.platform.startswith('linux') == True then the user
is running a Linux based operating system. The appropriate
if statement is then executed.

Otherwise if sys.platform.startswith('win') == True then
the user is running a windows based operating system.
The appropriate if statements are then executed.

Once we've entered the specific relative if statement.
We call the function self.getProperty('voices') wish
contains a list of all the available tts voice objects.
This list is returned to the variable voices.

We then call the function
self.setProperty('voice', + string or list). You can
manipulate this setting in a variety of ways. with a
string with a list element etc.. Once this has been
called and run the voice is set.
```

Definition at line 1810 of file Sentience.py.

Here is the call graph for this function: Here is the caller graph for this function:

6.5.3.36 set_speech_rate()

```
def Sentience.SentienceScreen.set_speech_rate (
              self )


    def set_speech_rate(self)

    Parameters:
    -----------
param1 : self
    Denotes it as being a member of
    SentienceScreen(Screen) class.

    Attributes
    ----------
rate
    The variable rate is used to store the integer
    value of the speech rate property belonging to
    self.engine. This rate determines the rate of
    words spoken per minute. I manually set this
    rate to rate - 40.

    Members
    -------
self.engine.getProperty('rate')
    We call this function to get the current rate of
    speech. This rate of speech is words per minute
    spoken. We store this value in the variable rate.

self.engine.setProperty('rate', rate-40)
    We call this function to set the rate of words per
    spoken per minute. The rate of words spoken per
    minute is set to current_rate - 40.

    Private Members
    ---------------
None

    Returns
    -------
return None
```

```
    Exceptions
    ----------
OSError
    The OSError can occur due to numerous reasons.
    What I'm primarily concerned with here however
    is import statements, incompatible Operating
    systems, and bad system calls. The exception
    if it occurs is handled and logged in an error
    log text file.

IOError
    The IOError can occur due to many reasons.
    My primary concern is file manipulation. The
    improper opening/closing/writing to files. If
    the exception occurs it's handled and logged; in
    an error log text file.

RunTimeError
    The RunTimeError error here is checking to make
    sure that the chat bot doesn't die. Essentially
    I just need to make sure that it completes and
    executes the python text to speech functions in
    a manner that doesn't cause a fatal exception. If
    something does occur the exception will be handled
    and logged to an error log text file.

ValueError
    Ensures that values passed to the chat bot are
    appropriate. And if for some reason one isn't
    the exception will be handled and logged to an
    error log text file.

    Notes
    -----
This function is called in SentienceScreen().__init__(self, **kwargs)
call. the purpose of this function is to change the default text to speech
rate of words spoken per minute.

When the function first runs we check to see what operating system the
user is running. If sys.platform.startswith('linux') == True then
the user is running a Linux based operating system. The appropriate if
statement is then executed.

Otherwise if sys.platform.startswith('win') == True then the user is
running a windows based operating system. The appropriate if statements
are then executed.

Once we've entered the specific relative if statement. We call the
function self.getProperty('rate') returns the current rate and stores it
in the variable rate.

We then call the function self.setProperty('rate', integer_value).
You can manipulate this setting in a variety of ways. You can
either set the integer parameter with an integer variable. Or
preform a mathematical operation on the current_rate like I have.
```

Definition at line 1950 of file Sentience.py.

Here is the call graph for this function:

6.5.3.37 set_tooltip_text()

```
def Sentience.SentienceScreen.set_tooltip_text (
              self,
              text )
```

```
set_tooltip_text(self, text)

Parameters
----------
    param1 : self
self denotes this function as being a member of
SentienceScreen().

    param2 : text
text is a string variable which holds a string
passed to it by the developer. In this case,
the string contains descriptive text about
each specific button on the menu bar (Action
Bar). It's called from within the self.on_mouse_pos()
function; and relates to each specific position. In
other words this function is called and each time
the "text" parameter contains different text for
each different button.
Attributes
----------
    self.tooltip
Refers to the ToolTipLabel in the kv design
language. This is the mutable instantiated
object of that widget. We use this to add/remove
the widget to and from the screen. As well as
changing its text. We can also do whatever else
to the widget that's possible with this object.

Members
-------
    self.tooltip.text
The way that we change the text of the label,
tooltip is both a function and a property.
It's a property and it's set but it's set by
a function call. We set the text of tooltip
by saying self.tooltip.text = 'insert text'.
We use this property to set and change the
text for each button on the menu bar (Action
Bar).
Private Members
---------------
    None
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
    This function is really straight forward.
    Every time the user hovers his or her pointer
    (mouse) over a button on the menu bar (Action
    Bar) a tooltip is created and added to the screen.
    Before the tooltip is added to the screen we change
    its text so that it contains information specific to
    the button that the mouse just touched.
```

Definition at line 4394 of file Sentience.py.

Here is the caller graph for this function:

6.5.3.38 set_volume()

```
def Sentience.SentienceScreen.set_volume (
              self,
              vol )
```

```
    Set the chatbots volume to vol. Zero is the lowesst
    value and one is the highest possible volume. Setting
    this to 1 will cause your audio device to produce
    failures in the sound in the form of crackling and
    or static.
```

Definition at line 1056 of file Sentience.py.

6.5.3.39 start_get_response_thread()

```
def Sentience.SentienceScreen.start_get_response_thread (
              self )


self.start_get_response_thread(self)

Parameters
----------
    param1 : self

Attributes
----------
    target
target is a member of the Process() class.
Members
-------
    self.ids.notification_widget.text
       This is a member of SentienceScreen() and is the
       notification_widget TextInput Widget in the kv
       design language. We use this to set the current text of the
       notification_widget TextInput Widget to the supplied string.
       This widget is used to display the text '..Thinking..'
       while the chatbot is locating an appropriate response for
       the user.
    self.ids.notification_widget.foreground_color
This is a member of SentienceScreen() and is the
notification_widget TextInput Widget in the kv
design language. We use this to set the current
text color of the widget. Here we see two examples.
If we change the text to ...Thinking... we change the
color to red; this shows the user that the program is
active and that the chatbot is currently generating a
response. If we change the text to ...Inactive... then
we change the text color to blue. This tells the user
that the chatbot has generated a response and that
the user can once again speak to it.
    Threading.Thread(target = event, args = (tuple)).start()
threading.Thread() is called to create a new thread.
This is used to run the acquisition of the a response
from the chatbot to the user. We pass the
function self.get_caprica_response to this as its event
and because self.get_caprica_response doesn't require
any arguements we omit the "args" parameter that we
saw in the threading code. We then call start() to
actually run the thread.
    Time.sleep(time_interval)
We call this function to force the program to "sleep",
for 1 second. This executes the two lines above
Time.sleep(1) before it creates and starts the thread.
We do this to prevent hanging issues caused by the
threading event and to ensure that the proper text
and color of the text is set; and more importantly,
so that the user can see this text.
    kivy.utils.get_color_from_hex('Hex string')
This is a kivy function. It's a member of kivy.utilsself.
We call this function to convert a hexadecimal string
into an equivelant opengl based rbga color.
```

```
    sys.platform.os.startswith('string')
This is a member of the Python sys (system) library.
We call this function to check the major version
of the users operating system. By doing so we can
determine if the user is running a windows or linux based
operating system. Note: See how I said "A" instead of
windows 10 or ubuntu? By using .startswith('') we
simply detect the operating system and are able
to be truly cross platform.
    time.sleep()
This is a member of the Time class(). We call this
function to force the program to sleep for one second.
To ensure that the notification_widget TextInput
text property reflects the current state of the program.
Private Members
---------------
    self.__set_thinking_text
We call this function to inform the user that the
chatbot is about to generate a response. The
self.notification_widget has its text set to
'...Thinking...' and the text is also made red.
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
    This function is called when the user sends a response to
    the chatbot. We use this to start self.get_caprica_response
    as a new thread to improve performance.

    We first detect the operating system, then set the text
    of self.notification_widget to be appropriate, we then
    change the color to reflect the text as mentioned
    above. We then force the program to sleep for one second
    to ensure those changes take effect. We finally create and
    run a new thread which then starts the
    self.get_caprica_response() function.
```

Definition at line 5337 of file Sentience.py.

Here is the call graph for this function:

6.5.3.40 start_timer_thread()

```
def Sentience.SentienceScreen.start_timer_thread (
              self,
              _time )


start_timer_thread(self, _time)

Parameters
----------
    self
self denotes this function as being a member of
SentienceScreen().
    _time
_time is a double variable which contains a number.
That numbers refers to a specific time value.
For instance, if we pass 20 to _time it means
five seconds. We use _time to run an event for _time
length.
Attributes
----------
self.ids.notification_widget
    Refers to the notification_widget TextInput Widget in
```

```
    the kv design language.

target
   target is an attribute of the threading.Thread class.
   We use that to register our event, which in this case
   is the function self.caprica_timer.
args
    args is an attribute of the threading.Thread class.
    Its a tupple of arguments which will store the parameters
    of the event that target =. In this case args = _time
    which again holds a numerical value which refers to the time
    that the function self.caprica_timer will run. To be more
    accurate it's the time that self.caprica_timer will count
    down from.
Members
-------
    self.notification_widget.opacity
This both a function and a property. We use
this to set the opacity of the notification_widget
TextInput widget which is in the kv design language.
When opacity = 1 it's visible to the user. When
opactiy = 0 it's invisible to the user.

    threading.Thread()
Thread is a member of the threading class. We use this
to decalre, initialize and run a new thread.
    threading.Thread.start()
start() is a member of the threading.Thread class. This
is what we use to actually start or run our newly
created thread. Which in this case is
self.caprica_timer().
Private Members
---------------
    None
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
    We call this function to start a new thread to run
    the function self.caprica_timer. It's run as a seperate
    thread to prevent the user from thinking that the program
    is crashing. It's also much more efficent to do it this way.

    Unfortunately, on windows operating systems threading and
    multiprocessing has the effect of launching a new python
    interpreter in the form of a new window which quickly pops
    up and vanishes from the screen which could cuase fear in the
    user.

    However, this is not a bug, it's an intended feature. Python
    is neither meant for nor truly not meant for multithreading.
    However, the GIL or Global Interpereter Lock prevents true
    multithreading from occuring to prevent huge memory leaks
    and unsafe practices. There is unfortunately no way around
    this windowing effect. But, it's okay because aside from it
    being a minor annoyance it's not an actual issue.

    Essentially, this function is called and it sets the opacity
    of notification_widget TextInput Widget to 1; rendering it
    visible to the user. A new thread is then created and executed
    which enables the notification_widget to display "..Thinking.."
    while the bot searches its database for an answer.

    Users may or may not see this notification based on the
    "Magic Window" that pops up and based on the amount of time
    that it takes the bot to locate an appropriate response.
```

Definition at line 4577 of file Sentience.py.

Here is the call graph for this function:

6.5.3.41 start_voice_response_thread()

```
def Sentience.SentienceScreen.start_voice_response_thread (
              self )


start_voice_response_thread(self)

Parameters
----------
    param1 : self
Denotes this as being a member of the SentienceScreen()
class.
Attributes
----------
    self.voice_disabled
This is a member of the SentienceScreen() class.
We use this boolean variable asa flag o tell us
whether or not the user enabled or disabled the
voice option. If the user has disabled the voice
they will informed that the voice option is
disabled and that they need to enable it. They
can do so by clicking on the red microphone
button on the menu bar (Action Bar).
    self.vocice_enabled
This is a member of the SentienceScreen() class. We
we use this boolean variable as flag to tel us
whether or not the user has enabled or disabled
the voice option. If the voice option is activated
the users microphone will be opened and voice input
will be recorded as long as the microphone picks up
noise.
    self.mic
self.mic is a member of the SentienceScreen() class.
It's also the instantiated object of sr.Microphone).
We use this object to open the users microphone if
they have one and pipe the input through sthe source
variable. When the user stops speaking the microphone
should close the audio in source which is being
listened to is then returned to the audio variable.
    source
source is a local variable of the function
self.get_caprica_voice_thread(). We use this variable
to store the input piped from the users microphone.
Once the microphone stops picking up audio input
we then transcribe the audio data into a string
by passing it to self.recognize_sphinx(audio).
    statement
statement is a local variable of the function
self.get_caprica_voice_thread() we use it to
store the transcribed string which is returned
to us by the self.recognize_sphinx(audio) function.
    audio
audio is a local variable of the function
self.get_caprica_voice_thread(). We use this
variable to store the audio data collected
by source which was piped through the users
microphone. We then pass this variable to
self.recognize_sphinx(audio). Which is then
transcribed from audio data and returned as
a string and stored in the statement variable.
    self.master_log
self.master_log is a member of the SentienceScreen()
class. This variable is used to store the full
conversation between the chatbot and the user. This
variable is later used to write data to a file.
Members
-------
    sys.platform.startswith(string)
```

```
This is a member of the sys() class. We call this
function to find out which operating system the
user is running. To be specific, we're only checking
for windows and linux based operating systems. This
funtion returns True if it matches 'linux', if
this happens we know that the user is running a
linux based operating system. If it returns False,
we then check to see if the user is running a windows
based operating sytem by passing 'win' to the function.
    self.record.listen(source)
We call this function to "listen" or, accept and store
the audio being piped through the users microphone
into the source variable. When the microphone no
longer detects audio input this audio data is
returned and stored in the audio variable.
    self.ids.view_port.text
This is a member of the SentienceScreen() class. We use
this to set the view_port TextInput widgets text field.
If the user hasn't enabled the voice option we
inform the user that the voice option is currently
disabled and that they can enable it by clicking
on the red mirophone button on the menu bar (Action
Bar).
    self.record.recognize_sphinx(audio)
We call this function to transcribed the passed
audio file into a string. The audio passed was
collected via the users microphone. This audio
data is transcribed into a string and returned and
stored in the statement variable.
    threading.Thread(*args)
This is a member of the threading() class. We use
this to declare, instantiate and run our thread all at
once. The thread is given a name based on the users
operating system. Ie, if it's linux, it's named
'linux_thread' and if it's windows it's named
'windows_thread'. We then pass it the target event
which is self.get_caprica_voice_thread(statement).
We finally call the start() function of the
threading class to actually start the new thread.
    datetime.datetime.now().strftime(string)
This is a member of the datetime() class.
We use this function to return the current local time
inside our file appending function. The time format
is set to year, month, day, hour, minute, seconds.
    self.get_caprica_voice_thread(string)
This is a member of the SentienceScreen() class.
We call this function as the event which is the
thread. That is to say this is the new thread. We
pass it the users transcribed verbal statement which
is stored in the string variable statement.
    time.sleep(integer)
This is a member of the time() class. We call
this function to put the program to sleep for one
second. We do this to ensure that the thread is
not executed until after the text, and the color of
the text in the self.notification_widget TextInput
widget have been changed to reflect the programs
current status.
Private Members
---------------
    self.__append_file(string, path)
This is a member of the SentienceScreen() class.
We call this function and pass it the string
which contains the users response to the chatbot
as well as the date and time that this response
occured. We then supply it with the absolute file
path of the file that we're writing to which is
the User_Statements.text file. This path depends on
the users operating system.
    self.__currently_thinking(bool)
This is a member of the SentienceScreen() class.
We call this function to set the current text and
color of that text to reflect the programs current
```

```
status. We pass it a boolean variable which is used
to determine this status. For more information on
this function see its comments.
Exceptions
----------
    None
Returns
-------
    None
Notes
-----
    This function is really straightforward. We use this
    when the user clicks on the record user button. Which is
    represented by the blue talking human head on the menu bar.

    If the button is red when the user clicks it, it means that
    the voice option wasn't enabled. We then inform the user
    in the view_port TextInput widget that the voice option
    is not currently enabled. We also inform them how to
    activate this option.

    Once the voice mode is activated and the record user
    button has been clicked. We open the users mirophone
    and pipe the audio input into source which is passed
    to the listen() function and stored in its audio form.

    When the microphone stops picking up audio input
    listen() function terminates and returns the audio
    to the local variable named audio.

    We then pass the audio variable into the function
    recognize_sphinx() which transcribes it into a string.
    Returns that string to be stored in the local variable
    named statement.

    We then append the users response to the chatbot as well
    as the date and time this response occured to the
    User_Statement text file. We then add this response to
    the end of the master_log string. Along with the users
    username.

    We then set the current status of the chatbot, Ie,
    thinking or inactive. Which is then reflected in
    the notification_widget text property. We then call
    time.sleep() and give it a one second interval to
    ensure that the above does occur before the thread
    is setup and run.
```

Definition at line 5452 of file Sentience.py.

Here is the call graph for this function:

6.5.3.42 write_logs()

```
def Sentience.SentienceScreen.write_logs (
              self )


def write_logs(self)
    This function is called when the user clicks the
    'Write Logs' button that's located on the menu bar.
    It's represented by the pencil. The purpose of this
    function is to write the contents of self.master_log
    and self.__user_profile to a text file named after
    the current user.

Parameters:
-----------
    param1 : self
```

```
Denotes it as being a member of SentienceScreen(Screen)
class.

Attributes
----------
    self.master_log
We write the contents of self.master_log to a text
file named after the current user. We also write
the contents of self.__user_profile to the text file.

    self.username
The self.username variable stores the users
input username. We use this variable to name
the file generated by this function.
self.username + '_Conversation.txt'

Members
-------
    os.path.isfile('path to file')
We call this function to ensure that
the files we're attempting to manipulate
don't already exist. If os.path.isfile() == True
then the file exists and will be over written.
If os.path.isfile() == False then
the file does not exist and we will write
the file normally.

    self.create_dir(self, path)
We call this function to check to make sure
that the folder holding the required files
for this program already exists. If it does exist we
skip this if statement and write the file created by
this function. If it doesn't exist we call
self.create_dir(path) and re-create the folder so
that we can store the soon to be created file.

Private Members
---------------
    self.__create_files(self, path)
This function is called only if one
of the files required files has been deleted.
This function will then write the file to
the disk.

    self.__user_profile
The dictionary variable self.__user_profile
contains a series of keys, Username, Sex, and
gender. This information is written to the start
of the file created by this function to clearly
state in text who the user is.

Returns
-------
    return None

Exceptions
----------
    OSError
The OSError can occur due to numerous reasons.
What I'm primarily concerned with here however
is import statements, incompatible Operating
systems, and bad system calls. The exception
if it occurs is handled and logged in an error
log text file.

    IOError
The IOError can occur due to many reasons.
My primary concern is file manipulation. The
improper opening/closing/writing to files. If
the exception occurs it's handled and logged; in
an error log text file.

    RunTimeError
```

```
The RunTimeError error here is checking to make sure
that the chat bot doesn't die. Essentially I just need
to make sure that it completes and executes the python
text to speech functions in a manner that doesn't cause
a fatal exception. If something does occur the exception
will be handled and logged to an error log text file.

    ValueError
Ensures that values passed to the chat bot are
appropriate. And if for some reason one isn't the
exception will be handled and logged to an error
log text file.

    FileNotFoundError
This can occur in a variety of ways however my
primary concern is that file path the user selected
is broken. Resulting in an File Not Found error. If this
occurs it's handled and logged to an error file text log.

    NameError
Again this can occur in a variety of ways but the
primary concern is that the conversion to bytes does not
take place or breaks some how due to wacky Unicode
characters. In which case the exception is handled
and logged to an error log text file.
Notes
-----
    This function is called when the user clicks the
    'Write Logs' button that's located on the menu bar.
    It's represented by the pencil. The purpose of this
    function is to write the contents of self.master_log
    and self.__user_profile to a text file named after
    the current user.

    The first thing that we do when this function is
    check the users operating system. If
    sys.platform.startswith('linux') == True
    then the user is running a linux based operating
    system and the appropriate if statements are
    executed.

    Other wise if sys.platform.startswith('win') == False
    then the user is running a windows based operating system
    and the appropriate if statements are executed.

    We then ensure that the directory created when the program
    first started exists. If it does not we re-create it. We
    then have to re-create the files that were stored in that
    folder.

    After that we create a new file naemd after the current
    user self.username + '_Conversation.txt'. We then write
    the contents of self.__user_profile and self.master_log
    to that file.
```

Definition at line 3785 of file Sentience.py.

Here is the call graph for this function:

6.5.4 Member Data Documentation

6.5.4.1 __is_thinking

```
Sentience.SentienceScreen.__is_thinking  [private]
```

Definition at line 971 of file Sentience.py.

6.5.4.2 _popup

`Sentience.SentienceScreen._popup [private]`

Definition at line 3989 of file Sentience.py.

6.5.4.3 audio_disabled

`Sentience.SentienceScreen.audio_disabled`

Definition at line 998 of file Sentience.py.

6.5.4.4 audio_enabled

`Sentience.SentienceScreen.audio_enabled`

Definition at line 997 of file Sentience.py.

6.5.4.5 audio_threshold

`Sentience.SentienceScreen.audio_threshold`

Definition at line 991 of file Sentience.py.

6.5.4.6 chatbot

`Sentience.SentienceScreen.chatbot`

Definition at line 981 of file Sentience.py.

6.5.4.7 current_conversation

`Sentience.SentienceScreen.current_conversation`

Definition at line 1011 of file Sentience.py.

6.5.4.8 engine

`Sentience.SentienceScreen.engine`

Definition at line 976 of file Sentience.py.

6.5.4.9 master_log

`Sentience.SentienceScreen.master_log`

Definition at line 993 of file Sentience.py.

6.5.4.10 mic

`Sentience.SentienceScreen.mic`

Definition at line 980 of file Sentience.py.

6.5.4.11 record

`Sentience.SentienceScreen.record`

Definition at line 979 of file Sentience.py.

6.5.4.12 tooltip

`Sentience.SentienceScreen.tooltip`

Definition at line 973 of file Sentience.py.

6.5.4.13 tooltip_open

`Sentience.SentienceScreen.tooltip_open`

Definition at line 972 of file Sentience.py.

6.5.4.14 user_input

`Sentience.SentienceScreen.user_input`

Definition at line 996 of file Sentience.py.

6.5.4.15 user_profile

`Sentience.SentienceScreen.user_profile`

Definition at line 999 of file Sentience.py.

6.5.4.16 username

```
Sentience.SentienceScreen.username
```

Definition at line 1000 of file Sentience.py.

6.5.4.17 voice_disabled

```
Sentience.SentienceScreen.voice_disabled
```

Definition at line 995 of file Sentience.py.

6.5.4.18 voice_enabled

```
Sentience.SentienceScreen.voice_enabled
```

Definition at line 994 of file Sentience.py.

The documentation for this class was generated from the following file:

- Sentience.py

6.6 Sentience.sentienceScreenManager Class Reference

Inheritance diagram for Sentience.sentienceScreenManager:

Collaboration diagram for Sentience.sentienceScreenManager:

6.6.1 Detailed Description

```
sentienceScreenManager(ScreenManager)
container/manager for SentienceScreen:
```

Definition at line 6186 of file Sentience.py.

The documentation for this class was generated from the following file:

- Sentience.py

7 File Documentation

7.1 Sentience.py File Reference

Classes

- class Sentience.PrintDialog
- class Sentience.DeleteDialog
- class Sentience.SentienceScreen
- class Sentience.ActionInput
- class Sentience.sentienceScreenManager
- class Sentience.SentienceApp

Namespaces

- Sentience

Variables

- string Sentience.__author__ = 'Aaron Johnson'
- string Sentience.__copyright__ = 'Copyright (c) 2018 Copyright Holder All Rights Reserved.'
- string Sentience.__license__ = 'MIT'
- string Sentience.__version__ = '2.1'
- string Sentience.__maintainer__ = 'Aaron Johnson'
- string Sentience.__email__ = 'Aaronjohnson@protonmail.ch'
- Sentience.root_widget = Builder.load_string()

7.2 Sentience.py

```
00001 import sys
00002 import os
00003 import subprocess
00004 import datetime
00005 import time
00006 import threading
00007 import speech_recognition as sr
00008 import pyttsx3
00009 from chatterbot import ChatBot
00010 import shutil
00011 import cProfile
00012 from kivy.uix.label import Label
00013 from kivy.uix.textinput import TextInput
00014 from kivy.uix.button import Button
00015 from kivy.core.window import Window
00016 from kivy.app import App
00017 from kivy.lang import Builder
00018 from kivy.clock import Clock
00019 from kivy.uix.screenmanager import ScreenManager, Screen
00020 from kivy.factory import Factory
00021 from kivy.uix.actionbar import ActionItem, ActionButton
00022 from kivy.config import Config
00023 from kivy.uix.floatlayout import FloatLayout
00024 from kivy.uix.popup import Popup
00025 from kivy.properties import ObjectProperty, StringProperty, ListProperty, ConfigParserProperty
00026 import kivy.utils
00027 from kivy.config import ConfigParser
00028 from kivy.uix.settings import SettingsWithSidebar
00029 from SettingsMenu import my_settings
00030
00031
00032
00033 __author__ = 'Aaron Johnson'
00034 __copyright__ = 'Copyright (c) 2018 Copyright Holder All Rights Reserved.'
```

```
00035 __license__ = 'MIT'
00036 __version__ = '2.1'
00037 __maintainer__ = 'Aaron Johnson'
00038 __email__ = 'Aaronjohnson@protonmail.ch'
00039
00040
00041 class PrintDialog(FloatLayout):
00042     '''
00043     PrintDialog(FloatLayout):
00044
00045     Parameters
00046     ----------
00047     param1 : FloatLayout
00048         The first parameter. Will hold the widgets in the Popup window which
00049         creates a PrinterDialog window. Allowing the user to navigate to and
00050         select a file for printing.
00051
00052     Attributes
00053     ----------
00054     print_files = ObjectProperty(None)
00055         print_files binds to the SentienceScreen().print_files() function.
00056
00057     Cancel = ObjectProperty(None)
00058         Cancel binds to the SentienceScreen().dissmis_popup() function.
00059
00060     Members
00061     -------
00062         None
00063     Private Members
00064     ---------------
00065         None
00066     Exceptions
00067     ----------
00068         None
00069     Returns
00070     -------
00071         None
00072     Notes
00073     -----
00074     This class is essentially a container for the Popup() that's created
00075     in SentienceScreen() class. The purpose of the Popup() is to allow
00076     the user to have a graphical window to navigate to, and select from,
00077     a list of files that they want to print out. Rather than automatically
00078     printing out the files for the user. This prevents potential issues
00079     and also allows the user the freedom to print out different files
00080     created by this program.
00081     '''
00082     print_files = ObjectProperty(None)
00083     Cancel = ObjectProperty(None)
00084
00085
00086 class DeleteDialog(FloatLayout):
00087     '''
00088     DeleteDialog(FloatLayout):
00089
00090     Parameters
00091     ----------
00092         param1 : FloatLayout
00093             This is pretty much exactly what it looks like. When this
00094             is used later on it will automatically add a float layout.
00095     Attributes
00096     ----------
00097         delete_file
00098             This will be used along side an ObjectProperty to register
00099             it for use with the SentienceScreen.delete_file() and
00100             SentienceScreen().open_delete_file_dialog() functions.
00101
00102         Cancel
00103             This will be used along side an ObjectProperty to register
00104             it for use with the SentienceScreen.dismiss_popup() function.
00105
00106         ObjectProperty(None)
00107             Initializes the two attributes to ObjectProperty. This is
00108             a built in feature of kivy to reduce code and make it
00109             easier to create/manipulate/initialize/instantiate
00110             both variables and functions. By making these two
00111             attributes object properties, in this case, we're
00112             literally binding them to the two functions calls
00113             listed above.
00114     Members
00115     -------
00116         None
00117     Private Members
00118     ---------------
00119         None
00120     Exceptions
00121     ----------
```

```
00122         None
00123     Returns
00124     -------
00125         None
00126     Notes
00127     -----
00128         We use this with our popup window for deleting specific files.
00129         This is our dialog. A FloatLayout is provided by default and
00130         two other layouts are added to it in the kv design language.
00131
00132         The ObjectProperty delete_file refers to a SentienceScreen()
00133         function: SentienceScreen.open_delete_file_dialog(). Clicking
00134         the button "Delete File" calls the open_delete_file_dialog()
00135         function which then opens a popup window.
00136
00137         The ObjectProperty Cancel refers to the the button "Cancel"
00138         which is contained in the above mentioned popup window.
00139     '''
00140     delete_file = ObjectProperty(None)
00141     Cancel = ObjectProperty(None)
00142
00143
00144 class SentienceScreen(Screen):
00145     '''
00146     SentienceScreen(Screen):
00147
00148     Parameters
00149     ----------
00150         param1 : Screen
00151             The first parameter creates a new Screen, which will function
00152             as a "page". This page is our only "Screen". It's the Main
00153             Window. It does everything. Now the actual designer code is
00154             done in the kv design language. But, this widget holds it
00155             all. It's the core of the program.
00156
00157     Attributes
00158     ----------
00159         self.chatbot
00160             The chatbot is the core feature here. It's the bot that the
00161             user communicates with. It's initialized and trained in the
00162             __init__ function. It's training can be continued throughout
00163             the program. Or expanded on by creating and adding new databases
00164             to its training regiment.
00165
00166         self.engine
00167             The engine object refers to the python3 text to speeh engine
00168             . It's what enables the chat bot to have a voice. From this
00169             engine we derive the ability to pass a string to the chat
00170             bot which can then access the systems text to speech software
00171             and read it back with an apropriate voice.
00172
00173         self.record
00174             The record object comes from speech_engine.Recognizer().
00175             This object allows us the ability to use programs such as
00176             CMU Sphinx voice recognition. Essentially we use this to
00177             transcribe recored audio to text which we can then store
00178             in a string. I make use of this by transcribing the recored
00179             audio to string vairables and passing them to the chat bot
00180             so that it can accurately respond to the user.
00181
00182         self.mic
00183             This object allows us to access and use any connected or
00184             onboard microphone if one is available. With this we can
00185             record a users voice, store it in a variable then send it
00186             to the Recognizer() to be transcribed and passed as a string
00187             to the chat bot.
00188
00189         self.audio_threshold
00190             This is used to automatically set the level at which the
00191             microphone accepts audio input. The higher the level the
00192             less sensitive the microphone is. Or rather the it's less
00193             likely that ambient noise will be treated as intentional
00194             audio being sent through the microphone.
00195
00196         self.record_dynamic_energy_threshold
00197             This applies to self.record and is a boolean variable. By
00198             setting this to False we can ensure that the energy_threshold
00199             doesn't dynamically set its energy_threshold level. Note:
00200             That the energy_threshold is what enables us to searate
00201             between ambient noise and the users intended voice commands.
00202
00203         self.master_log
00204             This is a string variable that I use to store all of the
00205             conversation that takes place between the user and the chat
00206             bot.
00207
00208         self.voice_enabled
```

```
00209             If self.voice_enabled is set to True then the user is able
00210             to use their microphone to communicate with the chat bot.
00211             Note: The user can only use a microphone if they have one.
00212             This can be either a connected microphone and or an onboard
00213             microphone.
00214
00215         self.voice_disabled
00216             If self.voice_disabled is set to True then the user can only
00217             communicate with the chat bot through text. Note: The chat
00218             bot can access its audio functions even if
00219             self.voice_disabled == True. This function only effects the
00220             users ability to use their microphone.
00221
00222         self.user_input
00223             This is a string variable which I use to store the input
00224             from the user the data here is passed to the chat bot,
00225             stored in various files and variables/data structures.
00226             Note: This variable is redundant and will in the future be
00227             removed. It can be ommited and replaced by the TextInput
00228             widgets return function.
00229
00230         self.audio_enabled
00231             if self.audio_enabled == True the chat bot can use the systems
00232             text to speech software (espeak, spai5, or nsss) to access the
00233             softwares built in voices and read back any strings that the
00234             chat bot comes up with as a response to the user. Note: This
00235             boolean vairable only effects the chat bots ability to use
00236             sound as a medium for communication. It does not effect the
00237             users ability to use their microphone.
00238
00239         self.audio_disabled
00240             If self.audio_disabled == True then the chat bot can only
00241             communicate with the user via text.
00242
00243         self.__user_profile
00244             self.__user_profile is a dictionary and stores three specific
00245             keys. 1) Username, 2) Age, 3) Gender. These are optional
00246             variables. The user doesn't need to create a user profile.
00247             Though it's encouraged that they do for better logging of
00248             the data. Note: If the user elects to not create a user profile
00249             this information is by default set.
00250
00251
00252     Members
00253     -------
00254         def __init__(self, **kwargs)
00255             Initalizes SentienceScreen() a more in depth analysis will
00256             be given under the SentienceScreen().__init__(self, **kwargs)
00257             functions documentation.
00258
00259         def quick_check_os(self)
00260             This function is called when the user clicks on the
00261             "Check Operating System" button which is represented by
00262             an image of a computer on the menu bar. This function
00263             when clicked checks to see if the user is running either
00264             windows or Linux. If the user is running windows it makes
00265             three new TextInput Widgets visible by changing the opacity.
00266             If the user is using a Linux operating system clicking on
00267             this button does nothing. A more in depth analysis will be
00268             given in the SentienceScreen().quick_check_os() functions
00269             documentation.
00270
00271         def get_user_text_response(self)
00272             This function is called when the user hits the "enter key"
00273             on their keyboard while inside of the user_input TextInput
00274             Widget. A string variable is returned from this and passed
00275             to the chat bot so that it can form a response to what the
00276             users statement was. A more in depth analysis of this will
00277             be given in the SentienceScreen().get_user_text_response()
00278             functions documentation.
00279
00280         def get_caprica_text_response(self)
00281             This function is called after the user inputs a text
00282             response. And that response is sent to the chat bot. The
00283             response that the user input is used by this function to
00284             generate a response from the chat bot. A more in depth
00285             analysis will be given in the
00286             SentienceScreen().get_caprica_text_response() functions
00287             documentation.
00288
00289         def get_user_voice_response(self)
00290             This function is called when the user clicks the
00291             "Record user" button. Which is located on the menu bar and
00292             is represented by the image of a blue talking head. If
00293             self.voice_disabled == True then the image will be a red
00294             talking head. If the user clicks the button when it's red a
00295             warning message will be displayed informing the user that
```

```
00296             he/she needs to first enable their microphone by clicking on
00297             the set_enable_disable_voice button. A More in depth
00298             analysis of this function will be given in the
00299             SentienceScreen().get_caprica_voice_response() function
00300             documentation.
00301
00302         def get_caprica_voice_response(self, words)
00303             This function is called after the user inputs a text string
00304             in the proper TextInput widget; or
00305             if self.voice_enabled == True. A more in depth analysis of
00306             this function will be given in the
00307             SentienceScreen().get_caprica_voice_response(self, words)
00308             function documentation.
00309
00310         def set_gender(self):
00311             This function is called in
00312             SentienceScreen().__init__(self, **kwargs). Through this
00313             function we set the voice property of self.engine to use
00314             the systems female voice option. A more in depth analysis
00315             of this function will be given in the
00316             SentienceScreen().set_gender(self) function documentation.
00317
00318         def set_speech_rate(self):
00319             This function is called in
00320             SentienceScreen().__init__(self, **kwargs). Through this
00321             function we can set the self.engine speech rate property.
00322             This function can in effect lower or increase the number
00323             of words spoken by the chat bot per minute. A more in
00324             depth analysis of this function will be given in the
00325             SentienceScreen().set_speech_rate() functions
00326             documentation.
00327
00328         def caprica_speak(self, words)
00329             This function is called from a variety of locations for the
00330             purpose of activating the voice feature of the chat bot
00331             which is derived from self.engine. A more in depth analysis
00332             of this function will be given in the
00333             SentienceScreen().caprica_speak(self, words) functions
00334             documentation.
00335
00336         def onEnd(self, name, completed)
00337             This function is called everytime
00338             self.caprica_speak(self, words) is called. This function is
00339             fired when the self.caprica_speak event has ended. This is a
00340             callabck which terminates the event queue of the
00341             self.engine. A more in depth analysis of this function will
00342             be given in the
00343             SentienceScreen().onEnd(self, name, completed) functions
00344             documentation.
00345
00346         def clear_viewport(self)
00347             This function is caleld whenever the user clicks the
00348             "Erase logs" button. Which is represented by the eraser on
00349             the menu bar. This button only erases the text in the
00350             viewport TextInput Widget. A more in depth analysis of
00351             this function will be given in
00352             SentienceScreen().clear_viewport(self) function
00353             documentation.
00354
00355         def create_user_profile(self)
00356             This function is highly redundant and will be removed in
00357             the future. This function is called when ever the user inputs
00358             their username for the first time. It runs some checks and
00359             then simply calls self.caprica_speak() to speak the users
00360             input username. A more in depth analysis of this function
00361             will be given in the
00362             SentienceScreen().create_user_profile(self) function
00363             documentation.
00364
00365         def set_enable_disable_audio(self)
00366             This function is called when the user clicks the
00367             self.set_enable_disable_audio button which is represented by
00368             either a red or blue speaker image on the menu bar. If
00369             self.audio_enabled == True the chat bot can use audio to
00370             communicate with the user and the image is a blue speaker.
00371             If self.audio_disabled == True then the chat bot can only
00372             communicate with the user via text. The button is  also
00373             then represented by a red speaker. This function will
00374             update the image on the menu bar to reflect its current
00375             status. A more in depth analysis of this function will
00376             be given in the SentienceScreen().set_enable_disable_audio(self)
00377             function documentation.
00378
00379         def set_enable_disable_voice(self)
00380             This function is called when the user clicks the
00381             self.set_enable_disable_voice button which is represented by
00382             either a red or blue microphone image on the menu bar.
```

```
00383             If self.voice_enabled == True the user can use their microphone
00384             to communicate with the chat bot and the image is a blue
00385             microphone. If self.voice_disabled == True then the user can
00386             only communicate with the chat bot via text. The button is
00387             also then represented by a red microphone. This function
00388             will update the image on the menu bar to reflect its current
00389             status. A more in depth analysis of this function will be
00390             given in the SentienceScreen().set_enable_disable_voice(self)
00391             function documentation.
00392
00393         def set_username(self)
00394             This function is called from two locations both involve the
00395             user inputting a desired username into a TextInput Widget
00396             and hitting the "Enter" key on their keyboard. This function
00397             sets the user name for the current user and can be changed
00398             at any time. A more in depth analysis of this function will
00399             be given in SentienceScreen().set_username() function
00400             documentation.
00401
00402         def set_sex(self)
00403             This function is called from two locations both involve the
00404             user inputting their gender into a TextInput Widget and
00405             hitting the "Enter" key on their keyboard. This function sets
00406             the gender for the current user and can be changed at any time.
00407             A more in depth analysis of this function will be given in
00408             SentienceScreen().set_sex() function documentation.
00409
00410         def set_age(self)
00411             This function is called from two locations both involve the
00412             user inputting their age into a TextInput Widget and hitting
00413             the "Enter" key on their keyboard. This function sets the users
00414             age for the current user and can be changed at any time. A more
00415             in depth analysis of this function will be given in
00416             SentienceScreen().set_username() function documentation.
00417
00418         def print_files(self, path, filename)
00419             This function is called when the user clicks on the "Print"
00420             button on the menu bar. When called a Popup() window is
00421             created and allows the user to navigate to any file that
00422             they wish to print. within that window are two buttons.
00423             Clicking the "Print" button will print the selected file
00424             while clicking the "Close" button will close the Popup()
00425             window. A more in depth analysis of this function will be
00426             given in SentienceScreen().print_files(self, path, filename)
00427             function documentation.
00428
00429         def create_dir(self, path)
00430             This function is caleld from within
00431             SentienceScreen().__init__(self, **kwargs). When executed it
00432             checks to see if a specific system relative directory exists.
00433             If it does the function returns nothing. If it doesn't exist
00434             the function creates the directory and then calls the private
00435             function self.__create_files(self, path). A more in depth
00436             analysis of this function will given in
00437             SentienceScreen().create_dir(self, path) function
00438             documentation.
00439
00440         def write_logs(self)
00441             This function is caleld when the user clicks the "Write Logs"
00442             button on the menu bar which is represented by a pencil
00443             image. It creates and writes the contents of self.master_log
00444             to a text file which is either
00445             "Users input username + _Conversations"
00446             .txt or simply "Username_Conversations".txt.
00447             A more in depth analysis of this function will be given in
00448             SentienceScreen().write_logs(self) function documentation.
00449
00450         def open_print_file_dialog(self)
00451             This function is caleld when the user clicks the "Print"
00452             button on the menu bar. This is the function that calls
00453             the Popup() window and allows the user to print a specific
00454             chosen file after navigating to it; and then by clicking the
00455             "Print files" button on that Popup() window.
00456
00457         def dismiss_popup(self)
00458             This function is called when the user clicks the "close"
00459             button on the PrintDialog() Popup() window. It closes the
00460             Popup() window. A more in depth analysis of this function
00461             will be given in the SentienceScreen().dismiss_popup()
00462             function documentation.
00463
00464         def on_mouse_pos(self, instance pos):
00465                 This function is called everytime that the user moves
00466                 his or her mouse. If the mouse collides with any of the
00467                 the buttons on the menu bar (Action Bar) this function
00468                 checks the positions against the various if statements
00469                 which relate to the specific button. When the position
```

```
00470                 of the users mouse matches the positions outlined in
00471                 the statements. A tool tip is displayed, which presents
00472                 at leat the name of the button.
00473
00474         def display_tooltip(self, *args):
00475             When this function is called the tooltip that relates
00476             to the button (as explain in on_mouse_pos) is created
00477             and added to the users screen. A clock event is then
00478             scheduled to delete the tooltip from the screen automaticaly
00479             after five seconds.
00480
00481         def close_tooltip(self, dt):
00482             This function is called by the clock event described in
00483             display_tooltip(). When this event is executed five seconds
00484             after it's been registered. The tooltip widget is
00485             deleted from the users screen.
00486
00487         def set_tooltip_text(self, text):
00488             We call this function and supply a string to the text
00489             parameter. This text relates to which ever button the users
00490             mouse colldied with. The text is then set and that's what's
00491             displayed to the user when the tooltip widget is added to
00492             the screen.
00493
00494         def caprica_timer(self, _time):
00495             This function is not currently in use. It's purpose
00496             was to function as an independent threaded timer. The time
00497             was based on the number supplied to the _time parameter.
00498             This function ticks down until _time is == 0 displaying
00499             the text ...Thinking... until _time is == 0; at which
00500             time the text displayed is then ...Inactive...
00501
00502         def start_timer_thread(self, _time):
00503             This function is not currently being used. But, it's
00504             purpose was to setup and run the caprica_timer function.
00505
00506         def check_timer(self, _time):
00507             This function is not being used. But, it's purpose was
00508             to check the status of self.caprica_timer(_time). To
00509             ensure that it ended when _time == 0 instead of counting
00510             down beyond that into negative numbers.
00511
00512         def get_caprica_response(self):
00513             This function is used to generate a response from the
00514             user. It combines all but the voice input/output
00515             responses. Basically, when you enter text into the
00516             user_input TextInput this function is called after
00517             the user hits the enter key. It then begins the
00518             process of the chatbot generating a response. It
00519             also runs as an independent thread.
00520
00521         def get_caprica_voice_thread(self, words):
00522             This function is called when the users has activated the
00523             voice option, then recorded their voice. Once that
00524             recording process is completed this function is called.
00525             This function then generates the chatbots response. It
00526             also runs as an independent thread.
00527
00528         def start_get_response_thread(self):
00529             We call this function after the user types some text
00530             into the self.ids.user_input TextInput widget, and
00531             then hits the enter key on their keyboard. This function
00532             changes the text of the notification_widget to
00533             '...Thinking...'. It then creates and runs the
00534             self.get_caprica_response() thread.
00535
00536         def start_voice_response_thread(self):
00537             We call this function after the voice option has been
00538             activated, and the user has hit the record button. Once
00539             the record button has been clicked, the user can begin
00540             speaking into their microphone. Once done speaking
00541             we create and run the self.get_caprica_voice_thread().
00542             # TODO: Fix notification text.
00543
00544         def _is_thread_stopped(self):
00545             We call this function to check if there are
00546             any active threads running.
00547             # TODO: This function is useless and should be removed.
00548
00549         def _stop_threading(self):
00550             This function is called when an active thread is
00551             supposed to be terminated. The idea is that the thread
00552             will be interupted and thus die.
00553             # TODO: Remove this because it doesn't do anything.
00554
00555         def get_user_text(self):
00556             This function is called to return the current
```

```
00557             text contained in the user_input TextInput widget.
00558
00559         def open_delete_file_dialog(self):
00560             This function is called when the users clicks on the
00561             delete file button which is located under the settings
00562             submenu on the menu bar. It opens a Popup() window. Which
00563             contains a filebrowser and allows the user to navigate to
00564             the file that they wish to delete. They can then select
00565             the file by clicking on it, and then clicking the delete
00566             button on the Popup() window. Or click the cancel button
00567             at any time which closes the window.
00568
00569         def delete_file(self, path, filename):
00570             This function is called after the user has slected a
00571             file in the Popup() window file browser and then clicked
00572             the delete button. The file the user selected is then
00573             deleted if it exists. If it doesn't exist the user is
00574             informed.
00575             # TODO: Remove path parameter as it does nothing at all.
00576
00577         def delete_all(self):
00578             We call this function if the user clicks on the
00579             **Delete All** button which is located in the
00580             settings submenu on the menu bar. Clicking this
00581             button deletes all files and folders generated by the
00582             this program. It also then exits the program.
00583
00584         def display_user_conversation(self):
00585             This function is called when the user clicks on
00586             the display conversation button. It outputs the
00587             contents of self.master_log into the view_port
00588             Widget.
00589
00590         def increase_chatbot_voume(self, vol):
00591             This function can be called to increase the volume
00592             of self.engine. The volume is increased by vol. The
00593             values it can take are between 0-1. With 0 being the
00594             lowest and one being the highest. # TODO: Re-implement
00595
00596         def decrease_chatbot_voume(self, vol):
00597             This function can be called to decrease the volume
00598             of self.engine. The volume is idecreased by vol. The
00599             values it can take are between 0-1. With 0 being the
00600             lowest and one being the highest. # TODO: Re-implement
00601
00602         def set_volume(self, vol):
00603             This function is called to set the volume of
00604             self.engine. The volume is set to vol; vol can be
00605             any  value between 0-1.
00606
00607         def increase_rate_of_speech(self, value):
00608             This funciton is called when the user increases
00609             the rate of speech using the settings menu. The
00610             current rate of self.engine is increased by value.
00611
00612         def decrease_rate_of_speech(self, value):
00613             This funciton is called when the user decreases
00614             the rate of speech using the settings menu. The
00615             current rate of self.engine is decreased by value.
00616
00617     Private Members
00618     ---------------
00619         def __create_files(self, path)
00620             This function is called from within the
00621             self.create_dir(self, path) function
00622             which is called first by the
00623             SentienceScreen().__init__(self, **kwargs) function.
00624             This function when called checks to see if specific files
00625             exist and if they don't
00626             it creates them. If they do already exist if essentially
00627             returns none. It's also called from one other function if a
00628             search does not find the required files which means that
00629             they were intentionally or unintentionally deleted. A more
00630             in depth analysis of this function will be given in
00631             SentienceScreen().__create_files(self, path) function
00632             documentation.
00633
00634         def __append_file(self, world, path)
00635             This function is caleld every time the user speaks to the
00636             chat bot and every time that the chat bot responds. The data
00637             passed to words is the response from both parties which is
00638             then appened to a specific file(s) which path comes from
00639             the path parameter. A more in depth analysis of this funciton
00640             will be given in
00641             SentienceScreen().__append_file(self, world, path) Note: The
00642             "World" param is a typo and needs to be changed to "word/words"
00643
```

```
00644         def __set_thinking_text(self, bool):
00645             This function is called to change the text and the
00646             color of the text of the notification_widget TextInput
00647             to reflect the current status of the program. Ie,
00648             if the chatbot is about to generate a response it
00649             says '...Thinking...' in red text. If the chatbot has
00650             already generated a response it says '...Inactive...'
00651             in blue text.
00652         def __currently_thinking(self, bool):
00653             This function is called to determine the current
00654             status of the program and the chatbot. If it's
00655             thinking or inactive.
00656             # TODO: This function is redundant
00657     Notes
00658     -----
00659         This is the essential widget. It's where everything happens.
00660     '''
00661
00662     def __init__(self, **kwargs):
00663         '''
00664         def __init__(self, **kwargs):
00665
00666         Parameters
00667         ----------
00668             param1 : self
00669                 Denotes this as being a member of the SentienceScree()
00670                 class.
00671
00672             param2 : **kwargs
00673                 **kwargs stands for keyword arguements. This
00674                 allows an arbitrary number of keyword arguements to
00675                 be passed to the self.SentienceScreen().__init__()
00676                 function.
00677         Attributes
00678         ----------
00679             mouse_pos
00680                 mouse_pos is an optional, though required for our
00681                 purposes, parameter of the Window.bind() function.
00682                 We call this function which is a member of the Window()
00683                 class. To register a mouse event. We bind the
00684                 traditional mouse_pos event to our own
00685                 self.on_mouse_pos(). The mouse (pointer) is always
00686                 tracked were simply binding it to one of our
00687                 functions so that we can monitor the position and
00688                 inctance of the pointer and call the bound function
00689                 when it's appropriate.
00690
00691             self.tooltip_open
00692                 self.tooltip_open is a member of the SentienceScreen()
00693                 class. We use this as a flag to determine whether or
00694                 not the ToolTipLabel widget is being shown.
00695
00696             self.mic
00697                 self.mic is a member of the SentienceScreen() class.
00698                 We use this to create our sr.Microphone() object.
00699                 This object allows us the ability to access and
00700                 manipulate the users microphone, assuming that
00701                 they have one. For later use in our program.
00702
00703             self.chatbot
00704                 self.chatbot is a member of the SentienceScreen()
00705                 class. We use this to create our ChatBot() object.
00706                 We can then manipulate self.chatbot, which we do,
00707                 throughout the rest of our program. This is one of
00708                 the core objects. Without this we have no chatbot.
00709
00710             self.audio_threshold
00711                 self.audio_threshold is a member of the
00712                 SentienceScreen() class. It stores an integer
00713                 value. This value enables us to force the users
00714                 microphone to ignore noises below a certain range.
00715
00716             self.audio_enabled
00717                 self.audio_enabled is a member of the SentienceScreen()
00718                 class. We use this boolean variable as a flag to tell
00719                 us if the user has enabled the audio option. The user
00720                 can enable the audio option by clicking on the red
00721                 speaker button on the menu bar (Action Bar). This
00722                 sets self.audio_enabled == True and changes the color
00723                 of the icon of the speaker button to blue.
00724
00725             self.audio_disabled
00726                 self.audio_disabled is a member of the SentienceScreen()
00727                 class. We use this boolean variable as a flag to tell
00728                 us if the user has disabled the audio option. The user
00729                 can disable the audio option by clicking on the blue
00730                 speaker button on the menu bar (Action Bar). This
```

```
00731               sets self.audio_enabled == False and changes the color
00732               of the icon of the speaker button to red.
00733
00734           self.record.dynamic_energy_threshold
00735               We use this to prevent self.record from dynamically
00736               Ie, contantly, checking the and setting the
00737               energy_threshold of self.record. Idealy, this should
00738               be left as a dynamic process but because no one
00739               microphone was created equal. Things get annoying
00740               really fast. So I've simply set it to a static
00741               variable for windows operating systems. And
00742               dynamically set it once for linux operating
00743               systems.
00744
00745           self.master_log
00746               self.master_log is a member of the SentienceScreen()
00747               class. It's a string variable that we use to store
00748               the users conversation with the chatbot. Every time
00749               that the user and the chatbot say something. Their
00750               responses are added to this string. We use this
00751               string to write data to files.
00752
00753           self.voice_enabled
00754               self.voice_enabled is a member of the SentienceScreen()
00755               class. We use this boolean variable as a flag to tell
00756               us if the user has enabled the voice option. The user
00757               can enable the voice option by clicking on the red
00758               microphone button on the menu bar (Action Bar). This
00759               sets self.voice_enabled == True and changes the color
00760               of the icon of the microphone button to blue. It also
00761               sets self.voice_disabled == False.
00762
00763           self.voice_disabled
00764               self.voice_disabled is a member of the SentienceScreen()
00765               class. We use this boolean variable as a flag to tell
00766               us if the user has disabled the voice option. The user
00767               can disable the voice option by clicking on the blue
00768               microphone button on the menu bar (Action Bar). This
00769               sets self.voice_enabled == False and changes the color
00770               of the icon of the microphone button to red.
00771
00772           self.user_input
00773               self.user_input is a member of the SentienceScreen()
00774               class. We use this string variable to temporarily
00775               store the contents of self.ids.user_input.text. Which
00776               is the TextInput widget that contains the users text
00777               comment to the chat bot. The data is returned to
00778               self.user_input when the user enters some text and hits
00779               the enter key on their keyboard while in the TexTInput
00780               widget.
00781
00782           self.__user_profile
00783               self.user_profile is a member of the SentienceScreen()
00784               class. We use this dictionary data structure to store
00785               the users information if they choose to give it. It
00786               stores their desired username, age, and gender. It's
00787               not a required thing. It's optional but personalizes
00788               a few things and helps to maintain more efficient logs
00789               of the conversations that the chatbots has. If there
00790               are multiple people speaking to it.
00791
00792           self.username
00793               self.username is a member of the SentienceScreen()
00794               class. We use this string variable to store the users
00795               desired username. Or, if the user elects not to supply
00796               a username we give this a default value of 'User: '
00797               and display it in the view_port TextInput widget
00798               to display the current conversation to the user.
00799
00800           self.on_mouse_pos
00801               self.on_mouse_pos is a member of the SentienceScreen()
00802               class. It's a function that we use to track and handle
00803               mouse events. If the user hovers their mouse over a
00804               button on the Menu bar (Action Bar). This function is
00805               called, which locates teh mouses position and instance
00806               of the mouse when it collided with a button. It then
00807               executes the appropriate if statements which then
00808               create a ToolTipLabel widget, change the text to
00809               reflect the button the user collided with. And then
00810               displays that label as a tooltip over the button.
00811
00812           self.engine
00813               self.engine is a member of the SentienceScreen()
00814               class. We use this to create our object of the pyttsx3
00815               class. This allows us to access the users systems text
00816               to speech software so that the response generated by
00817               the chatbot can be verbally delivered to the user. If
```

```
00818                 they elected to active either the audio or voice
00819                 options.
00820
00821             self.record()
00822                 self.record() is a member of the SentienceScreen()
00823                 class. It's the object of the sr.Recognizer()
00824                 class. This allows us to accept, transcribe and later
00825                 manipulate an audio recording of the user. This
00826                 occurs when the user has activated the voice option.
00827
00828             self.__is_thinking
00829                 self.__is_thinking is a member of the SentienceScreen()
00830                 class. We use this boolean variable as a flag to tell
00831                 us whether or not the chatbot is preparing to generate
00832                 a response for the user. Or has just finished generating
00833                 a response to the user. When the chatbot is generating
00834                 a response the text of the notification_widget is set
00835                 to '...Thinking...' and the color of that text is red.
00836                 When the chatbot finishes generating a response and has
00837                 sent it to the user the text is set to '...Inactive...'
00838                 and is blue.
00839
00840             self.current_conversation
00841                 This is a member of the SentienceScreen() class.
00842                 We use this string variable to store the current
00843                 contents of the view_port Widget. When a tooltip
00844                 is displayed. We do this to prevent the loss of
00845                 the information that was previously being displayed.
00846         Members
00847         -------
00848             super(SentienceScreen, self).__init__(**kwargs)
00849                 Here we're calling super dynamically to allow the
00850                 use of inheritance. This applies to the
00851                 sentienceScreenManager() class. It allows us to
00852                 work with the various widgets and screens.
00853
00854             Window.bind(mouse_pos = self.on_mouse_pos)
00855                 We call Window.bind() to bind the base Window
00856                 classes mouse_pos event to our mouse event. Which
00857                 in this case is self.on_mouse_pos()
00858
00859             threading.Event()
00860                 threading.Event() is a member of the threading()
00861                 class. We use this to create a threading event
00862                 which we'll use to interupt active threads later on.
00863
00864             Factory.ToolTipLabel(text = (string))
00865                 We use this to register and instantiate
00866                 classes anywhere anytime. In our case though
00867                 we're just setting this up and setting the text
00868                 field to '', Ie, an empty string.
00869
00870             Config.set('input', 'mouse', 'mouse', disable_multitouch)
00871                 Config is a member of the kivy base class. We call
00872                 this in our SentienceScreen.__init__() method
00873                 to disable kivys multitouch ability. This shuts off
00874                 users ability to interact via touch screen on touch
00875                 screen capable systems.
00876
00877             sys.platform.startswith(string)
00878                 This is a member of the sys() class. We call this
00879                 function to dertmine whether what operating system
00880                 the user is using. It returns a boolean value, if
00881                 the version matches either 'linux' or 'win'.
00882
00883             pyttsx3.init(string)
00884                 This is a member of the pyttsx3() class. We call
00885                 this function when we declare and instantiate
00886                 our object of this class. It also serves to
00887                 set the driver for the systems text to speech
00888                 software based on the users operating system.
00889
00890             sr.Recognizer()
00891                 This is a member of the speech_recognition() class.
00892                 We call this when we declare and instance our
00893                 self.record object. Which then allows us to
00894                 accept user input from a microphone and then
00895                 transcribe that uadio respone as a string for
00896                 later manipulation.
00897
00898             sr.Microphone()
00899                 This is a member of the speech_recognition() class.
00900                 We call this when we declare and instantiate our
00901                 self.mic object. Which then allows us to manipulate
00902                 the users microphone if they have one.
00903
00904             ChatBot()
```

```
00905                 Here we setup the ChatBot. We do wo when we decalre
00906                 and instantiate our self.chatbot object. We create and
00907                 supply the required filters and adapters which dictate
00908                 how this chatbot will learn.
00909
00910             self.set_gender()
00911                 This is a member of the SentienceScreen() class. We
00912                 call this function to set the gener of self.engine
00913                 to a female. This has the effect of changing the
00914                 default voice from a male, to female voice.
00915
00916             self.set_speech_rate()
00917                 This is a member of the SentienceScreen() class. We
00918                 call this function to set the speech rate of the
00919                 users systems speech to text software. In our
00920                 case we lower it so that when self.caprica_speak()
00921                 is called the resulting spoken string is done
00922                 so in a manner that the user can understand.
00923
00924             leng()
00925                 We call the built in python leng() or length
00926                 function to dertmine the length of self.username.
00927                 If the length is less than or equal to zero we
00928                 supply self.username with the default value of
00929                 'User: '. If the user elects later on to set their
00930                 own username then the self.user_profile overrides
00931                 this variable.
00932
00933             self.create_dir(path)
00934                 This is a member of the SentienceScreen() class.
00935                 We call this function to create a series of files
00936                 and folders that the user needs to operate
00937                 this program.
00938
00939             self.engine.connect(string, event)
00940                 We call this function to bind our events
00941                 to the pyttsx3 events. We connect self.onEnd
00942                 to the pyttsx3 'finished-utterance' event. This
00943                 event is fired when the pyttsx3 finishes speaking
00944                 whatever string was supplied to it. We also connect
00945                 self.caprica_speak to 'started-utterance' which is
00946                 fired when the systems text to speech software
00947                 begins speaking a supplied string.
00948         Private Members
00949         ---------------
00950             None
00951
00952         Exceptions
00953         ----------
00954             None
00955
00956         Returns
00957         -------
00958             None
00959
00960         Notes
00961         -----
00962             This is the initalization method of SentienceScreen().
00963             It's relatively comprehensive so I'm not going to explain
00964             it again. It's easy enough to understand whats happening
00965             when you reference the above comments.
00966         '''
00967
00968         super(SentienceScreen, self).__init__(**kwargs)
00969
00970         Window.bind(mouse_pos=self.on_mouse_pos)
00971         self.__is_thinking = False
00972         self.tooltip_open = False
00973         self.tooltip = Factory.ToolTipLabel(text='')
00974         Config.set('input', 'mouse', 'mouse, disable_multitouch')
00975         if sys.platform.startswith('linux'):
00976             self.engine = pyttsx3.init('espeak')
00977         elif sys.platform.startswith('win'):
00978             self.engine = pyttsx3.init()
00979         self.record = sr.Recognizer()
00980         self.mic = sr.Microphone()
00981         self.chatbot = ChatBot('Caprica',
00982                                storage_adapter='chatterbot.storage.SQLStorageAdapter',
00983                                logic_adapters=['chatterbot.logic.BestMatch', '
    chatterbot.logic.TimeLogicAdapter', 'chatterbot.logic.MathematicalEvaluation'],
00984                                input_adapter='chatterbot.input.VariableInputTypeAdapter',
00985                                output_adapter='chatterbot.output.OutputAdapter',
00986                                filters=["chatterbot.filters.RepetitiveResponseFilter"],
00987                                database='RC_2001-06.db',
00988                                trainer='chatterbot.trainers.ChatterBotCorpusTrainer')
00989         self.set_gender()
00990         self.set_speech_rate()
```

```
00991         self.audio_threshold = 400
00992         self.record.dynamic_energy_threshold = False
00993         self.master_log = str()
00994         self.voice_enabled = False
00995         self.voice_disabled = True
00996         self.user_input = str()
00997         self.audio_enabled = False
00998         self.audio_disabled = True
00999         self.user_profile = {1: 'Username', 2: 'Age', 3: 'Sex'}
01000         self.username = str()
01001         if len(self.username) <= 0:
01002             self.username = 'User'
01003         if sys.platform.startswith('linux'):
01004             self.create_dir('/home/' + str(os.getlogin()) + '/.SentienceFiles/')
01005         elif sys.platform.startswith('win'):
01006             self.create_dir('C://SentienceFiles//')
01007         elif sys.platform == 'darwin':
01008             self.create_dir('/home/' + str(os.getlogin()) + '/.SentienceFiles/')
01009         self.engine.connect('started-utterance', self.caprica_speak)
01010         self.engine.connect('finished-utterance', self.onEnd)
01011         self.current_conversation = str()
01012         self.set_volume(1)
01013
01014
01015
01016     def display_user_conversation(self):
01017         '''
01018             When called this function displays the contents of
01019             self.master_log inside the view_port TextInput Widget.
01020         '''
01021         self.ids.view_port.text = self.master_log
01022
01023
01024
01025     def increase_chatbot_volume(self, vol):
01026         '''
01027             Increase the chatbots volume by vol when called.
01028             Chatbots volume can only be set between 0-1
01029             with 0 being the lowest and 1 being the highest volume
01030             level.
01031
01032             If vol > 1 or < 0 a warning message is displayed in the
01033             view_port TextInput widget.
01034         '''
01035         if int(vol) >= 0 and int(vol) <= 1:
01036             volume = self.engine.getProperty('volume')
01037             self.engine.setProperty('volume', volume + vol)
01038         elif int(vol) < 0 or int(vol) > 1:
01039             self.ids.view_port.text = 'Value must be between 0-1.'
01040
01041
01042
01043     def decrease_chatbot_volume(self, vol):
01044         '''
01045             Decreases the chatbots volume by vol.
01046
01047         '''
01048         if int(vol) >= 0 and int(vol) <= 1:
01049             volume = self.engine.getProperty('volume')
01050             self.engine.setProperty('volume', volume - vol)
01051         elif int(vol) < 0 or int(vol) > 1:
01052             self.ids.view_port.text = 'Value must be between 0-1.'
01053
01054
01055
01056     def set_volume(self, vol):
01057         '''
01058             Set the chatbots volume to vol. Zero is the lowesst
01059             value and one is the highest possible volume. Setting
01060             this to 1 will cause your audio device to produce
01061             failures in the sound in the form of crackling and
01062             or static.
01063         '''
01064         volume = self.engine.getProperty('volume')
01065         self.engine.setProperty('volume', vol)
01066
01067
01068
01069     def get_user_text_response(self):
01070         '''
01071         def get_user_text_response(self)
01072
01073         Parameters:
01074         -----------
01075             param1 : self
01076                 Denotes it as being a member of SentienceScreen(Screen)
01077                 class.
```

```
01078
01079         Attributes
01080         ----------
01081             self.audio_disabled
01082                 Preforms a check to establish wether or not
01083                 self.audio_disabled == True.
01084
01085             self.user_input
01086                Accepts text (string) from user_input TextInput Widget.
01087
01088             self.ids.user_input
01089                 TextInput Widget that returns its current string to
01090                 self.user_input. Or self.get_caprica_voice_response()
01091
01092             self.master_log
01093                 Stores the contents of self.ids.user_input TextInput
01094                 string along with some other data.
01095
01096             self.audio_enabled
01097                 Checks to see if self.audio_enabled == True
01098
01099         Members
01100         -------
01101             self.get_caprica_voice_response(words)
01102                 Is called when self.audio_enabled == True Accepts
01103                 self.ids.user_input.text as its parameter. That is to
01104                 say that self.ids.user_input returns its string to this
01105                 function. The chat bot then searches its database and
01106                 locates the best  possible response. It then calls
01107                 self.caprica_speak() to read that response.
01108
01109         Private Members
01110         ---------------
01111             self.__append_file(path)
01112                 This is called to append the user_input value to the
01113                 User_Statements.txt file.
01114
01115         Returns
01116         -------
01117             return self.get_caprica_text_response()
01118                 Ends with the text response of caprica_speak() which is
01119                 a resposne to the users comment.
01120
01121             return None
01122                 Function returns None when self.audio_enabled == True
01123
01124         Exceptions
01125         ----------
01126         OSError
01127             The OSError can occur due to numerous reasons.
01128             What I'm primarily concerned with here however
01129             is import statements, incompatible Operating
01130             systems, and bad system calls. The exception
01131             if it occurs is handled and logged in an error
01132             log text file.
01133
01134         IOError
01135             The IOError can occur due to many reasons.
01136             My primary concern is file manipulation. The
01137             improper opening/closing/writing to files. If
01138             the exception occurs it's handled and logged; in
01139             an error log text file.
01140
01141         FileNotFoundError
01142             FileNotFoundError is exactly what it sounds like.
01143             If the file I'm trying to write to doesn't exist
01144             we may have a problem. But not to worry it's handled
01145             and logged.
01146
01147
01148         Notes
01149         -----
01150             **This function is deprecated and has been replaced
01151             with get_caprica_response**
01152
01153             This function is called when the user hits the "Enter" key
01154             on their key board while clicked into the
01155             self.ids.user_input TextInput Widget.
01156
01157             What happens next is determined by the value(s) of
01158             self.audio_enabled and self.audio_disabled.
01159
01160             if self.audio_enabled == True the chat bot obtains the users
01161             comment and then calls self.caprica_speak() where it
01162             essentially returns None
01163
01164             if self.audio_disabled == True the chat bot obtains the users
```

```
01165             comment passes that string to the
01166             self.get_caprica_text_response(string) and then returns
01167             self.get_caprica_text_response(string)
01168         '''
01169         try:
01170             if sys.platform.startswith('linux'):
01171                 if self.audio_disabled:
01172                     self.user_input = self.ids.user_input.text
01173                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + self.user_input, '/home/' + str(os.getlogin()) + '
      /.SentienceFiles/User_Statements.txt')
01174                     self.master_log += '\n' + self.username + ': ' + self.
      user_input
01175                     return self.get_caprica_text_response()
01176                 elif self.audio_enabled:
01177                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + self.ids.user_input.text, '/home/' + str(os.getlogin()) + '/.SentienceFiles/User_Statements.txt
      ')
01178                     self.master_log += '\n' + self.username + ': ' + self.ids.
      user_input.text
01179                     self.get_caprica_voice_response(self.ids.user_input.text)
01180             elif sys.platform.startswith('win'):
01181                 if self.audio_disabled:
01182                     self.user_input = self.ids.user_input.text
01183                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + self.user_input, 'C://SentienceFiles//User_Statements.txt')
01184                     self.master_log += '\n' + self.username + ': ' + self.ids.
      user_input.text
01185                     return self.get_caprica_text_response()
01186                 elif self.audio_enabled:
01187                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + self.ids.user_input.text, 'C://SentienceFiles//User_Statements.txt')
01188                     self.master_log += '\n' + self.username + ': ' + self.ids.
      user_input.text
01189                     self.get_caprica_voice_response(self.ids.user_input.text)
01190         except OSError as a:
01191             if sys.platform.startswith('linux'):
01192                 self.__append_file('\n' + 'Function: get_user_text_response ' + '\nOSError: '
      + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01193             elif sys.platform.startswith('win'):
01194                 self.__append_file('\n' + 'Function: get_user_text_response ' + '\nOSError: '
      + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01195             return None
01196         except IOError as b:
01197             if sys.platform.startswith('linux'):
01198                 self.__append_file('\n' + 'Function: get_user_text_response ' + '\nIOError: '
      + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01199             elif sys.platform.startswith('win'):
01200                 self.__append_file('\n' + 'Function: get_user_text_response ' + '\nIOError: '
      + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01201             return None
01202         except FileNotFoundError as c:
01203             if sys.platform.startswith('linux'):
01204                 self.__append_file('\n' + 'Function: get_user_text_response ' + '\n
      FileNotFoundError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/'
       + str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01205             elif sys.platform.startswith('win'):
01206                 self.__append_file('\n' + 'Function: get_user_text_response ' + '\n
      FileNotFoundError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01207             return None
01208
01209
01210
01211     def increase_rate_of_speech(self, value):
01212         '''
01213             Increases the words per minute spoken by value.
01214             Increasing this value increase teh rate of speech
01215             and may make the chatbots verbal messages incoherent.
01216         '''
01217         rate = self.engine.getProperty('rate')
01218         self.engine.setProperty('rate', rate + value)
01219
01220
01221
01222     def decrease_rate_of_speech(self, value):
01223         '''
01224             Decreases the chatbots words spoken per minute.
01225         '''
01226         rate = self.engine.getProperty('rate')
01227         self.engine.setProperty('rate', rate - value)
01228
01229
```

```
01230
01231     def get_caprica_text_response(self):
01232         '''
01233         def get_caprica_text_response(self)
01234
01235         Parameters:
01236         -----------
01237             param1 : self
01238                 Denotes it as being a member of SentienceScreen(Screen)
01239                 class.
01240
01241         Attributes
01242         ----------
01243             self.username
01244                 Contains the username that user entered if he/she entered
01245                 one. The length of self.username is checked to see if it's
01246                 less than or equal to zero. If it is it means that the user
01247                 never set a username and a default value of "User: " is set
01248                 This value along with the users response to the chat bot and
01249                 'Caprica: ' and then the chat bots response are displayed
01250                 in text in self.ids.view_port TextInput Widget.
01251
01252
01253             temp
01254                 temp is used to store the response from the chat bot
01255                 temporarily.
01256
01257             self.master_log
01258                 The string 'Caprica: ' and the value contained in temp are
01259                 added to the end of the string with a new line
01260                 character.
01261
01262             self.ids.view_port
01263                 This is the main view port TextInput Widget here we briefly
01264                 store the User text response and chat box text response.
01265
01266             self.ids.user_input
01267                 The string contained in the user_input TextInput Widget
01268                 is cleared and the hint_text reset.
01269
01270
01271         Members
01272         -------
01273             self.chatbot.get_response(self.user_input)
01274                 Obtains the response from the chat bot which is generated
01275                 to best fit the user response which is passed into this
01276                 function as a parameter. It then returns the chat bots
01277                 response and is in this case stored in the temp variable.
01278                 This is all type casted to str() to ensure type safety.
01279
01280         Private Members
01281         ---------------
01282             self.__append_file(self, words, path)
01283                 Is called to append the Chat bot responses to the text file
01284                 Caprica_Statements.txt
01285
01286
01287         Returns
01288         -------
01289             return None
01290
01291
01292         Exceptions
01293         ----------
01294         OSError
01295             The OSError can occur due to numerous reasons.
01296             What I'm primarily concerned with here however
01297             is import statements, incompatible Operating
01298             systems, and bad system calls. The exception
01299             if it occurs is handled and logged in an error
01300             log text file.
01301
01302         IOError
01303             The IOError can occur due to many reasons.
01304             My primary concern is file manipulation. The
01305             improper opening/closing/writing to files. If
01306             the exception occurs it's handled and logged; in
01307             an error log text file.
01308
01309         RunTimeError
01310             The RunTimeError error here is checking to make sure that
01311             the chat bot doesn't die. Essentially I just need to make
01312             sure that it completes and executes the python text to speech
01313             functions in a manner that doesn't cause a fatal exception. If
01314             something does occur the exception will be handled and logged to
01315             an error log text file.
01316
```

```
01317         ValueError
01318             Ensures that values passed to the chat bot are appropriate.
01319             And if for some reason one isn't the exception will be handled
01320             and logged to an error log text file.
01321 
01322 
01323         Notes
01324         -----
01325             **This function is deprecated and has been replaced
01326             with get_caprica_response**
01327 
01328             This function is called when self.audio_disabled == True
01329 
01330             It first checks the operating system.
01331             If sys.platform.startswith('linux') == True it executes
01332             the if statement intended for the linux operating system.
01333 
01334             Otherwise if sys.platform == False it executes the if
01335             statements intended for the windows operating system.
01336 
01337             It then enters the appropriate if statement
01338             and the user response that's stored in
01339             self.user_input is passed into
01340             self.chatbot.get_response(self.user_input) the generated
01341             response is then stored in the temp string variable.
01342 
01343             We then call self.__append_file('') we give it a new line
01344             character and the data stored in the temp variable and pass
01345             to the path parameter the apropriate path which is based off
01346             of the users operating system.
01347 
01348             We then set the string of the view_port TextInput Widget to be
01349             'Username: ' + user_response
01350             'Caprica: ' + caprica_response
01351 
01352             We then clear the seld.ids.user_input.text field
01353             (user_input TextInput Widget). So that the
01354             hint_text property is reset.
01355         '''
01356         try:
01357             if sys.platform.startswith('linux'):
01358                 if len(self.username) <= 0:
01359                     self.username = 'User'
01360                 temp = str(self.chatbot.get_response(self.user_input))
01361                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp, '/home/' + str(os.getlogin()) + '/.SentienceFiles/Caprica_Statements.txt')
01362                 self.master_log += '\nCaprica: ' + temp
01363                 self.ids.view_port.text = self.username + ': ' + self.
      user_input + '\nCaprica: ' + temp
01364                 self.ids.user_input.text = ''
01365             elif sys.platform == 'win32':
01366                 if len(self.username) <= 0:
01367                     self.username = 'User'
01368                 temp = str(self.chatbot.get_response(self.user_input))
01369                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp, 'C://SentienceFiles//Caprica_Statements.txt')
01370                 self.ids.view_port.text = self.username + ': ' + self.
      user_input + '\nCaprica: ' + temp
01371                 self.master_log += '\nCaprica: ' + temp
01372                 self.ids.user_input.text = ''
01373         except OSError as a:
01374             if sys.platform.startswith('linux'):
01375                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\nOSError:
       ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01376             elif sys.platform.startswith('win'):
01377                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\nOSError:
       ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01378                 return None
01379         except IOError as b:
01380             if sys.platform.startswith('linux'):
01381                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\nIOError:
       ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01382             elif sys.platform.startswith('win'):
01383                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\nIOError:
       ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01384                 return None
01385         except RuntimeError as c:
01386             if sys.platform.startswith('linux'):
01387                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\n
      RuntimeError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
       str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01388             elif sys.platform.startswith('win'):
01389                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\n
```

```
      RuntimeError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01390             return None
01391         except ValueError as d:
01392             if sys.platform.startswith('linux'):
01393                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\n
      ValueError: ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
      str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01394             elif sys.platform.startswith('win'):
01395                 self.__append_file('\n' + 'Function: get_caprica_text_response ' + '\n
      ValueError: ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01396             return None
01397
01398
01399
01400     def get_user_voice_response(self):
01401         '''
01402         def get_user_voice_response(self)
01403
01404         Parameters:
01405         -----------
01406             param1 : self
01407                 Denotes it as being a member of
01408                 SentienceScreen(Screen) class.
01409
01410         Attributes
01411         ----------
01412             self.voice_disabled
01413                 Checks to see if self.voice_disabled == True or
01414                 False If self.voice_disabled == True a warning
01415                 message is sent to the user in forming them that
01416                 they need to click on the red microphone image;
01417                 before clicking the "Record user" button. The
01418                 warning message is sent is displayed in
01419                 self.ids.view_port. If self.voice_disabled == False
01420                 self.voice_enabled == True and the user can begin
01421                 recording their voice via a microphone.
01422
01423             self.ids.view_port
01424                 A warning message is displayed in the view_port
01425                 TextInput Widget informing the user they need to
01426                 first enable the voice option before they can use
01427                 their microphone to speak with the chat bot.
01428
01429             self.mic
01430                 self.mic = sr.Microphone() this is what enables us to use
01431                 the microphone to speak with the chat bot.
01432
01433             source
01434                 When we begin recording the users voice we do so in a
01435                 loop we open that loop (open the microphone) as source.
01436                 All the audio detected is piped into source and then
01437                 stored in the audio variable.
01438
01439             audio
01440                 The recorded sound will be streamed to the function
01441                 self.record.listen(source) all audio picked up will
01442                 be saved in the audio variable for later transcription
01443                 into a string.
01444
01445             temp
01446                 The data contained in the audio variable is passed to
01447                 the function self.record.recognize_sphinx(source) which
01448                 will then transcribe the audio file and store the
01449                 returned string in the temp variable.
01450
01451             self.master_log
01452                 A new line character, the users username and the string
01453                 stored in the temp variable are then appended to the
01454                 self.master_log string.
01455
01456         Members
01457         -------
01458             self.record.listen(source)
01459                 This function which is derived from
01460                 speech_recognition.Recognizer()is called when we open
01461                 the loop for recording the users voice response. Source
01462                 is the stream for the microphone. The data contained in
01463                 source is piped into this function as its parameter and
01464                 then returned to the audio variable.
01465
01466             self.record.recognize_sphinx(audio)
01467                 The data contained in the audio variable is sent to this
01468                 function for transcription into a string. The data once
01469                 transcribed is returned to the temp variable for later
01470                 manipulation.
```

```
01471
01472             self.get_caprica_voice_response(string)
01473                 This function is called after the users voice response
01474                 has been transcribed and stored in the temp variable.
01475                 The temp variable is then passed into this function as
01476                 its parameter which then obtains the most accurate result
01477                 possible for the chat bot to respond to the user. This
01478                 response is then spoken by the self.caprica_speak()
01479                 function.
01480
01481         Private Members
01482         ---------------
01483             self.__append_file(string, path)
01484                 This function is called to append the users transcribed
01485                 voice response to the User_Statements.txt file along with
01486                 a new line character.
01487
01488         Returns
01489         -------
01490             return None
01491
01492
01493         Exceptions
01494         ----------
01495             OSError
01496                 The OSError can occur due to numerous reasons.
01497                 What I'm primarily concerned with here however
01498                 is import statements, incompatible Operating
01499                 systems, and bad system calls. The exception
01500                 if it occurs is handled and logged in an error
01501                 log text file.
01502
01503             IOError
01504                 The IOError can occur due to many reasons.
01505                 My primary concern is file manipulation. The
01506                 improper opening/closing/writing to files. If
01507                 the exception occurs it's handled and logged; in
01508                 an error log text file.
01509
01510             RunTimeError
01511                 The RunTimeError error here is checking to make sure
01512                 that the chat bot doesn't die. Essentially I just need
01513                 to make sure that it completes and executes the python
01514                 text to speech functions in a manner that doesn't cause
01515                 a fatal exception. If something does occur the exception
01516                 will be handled and logged to an error log text file.
01517
01518             ValueError
01519                 Ensures that values passed to the chat bot are
01520                 appropriate. And if for some reason one isn't the exception
01521                 will be handled and logged to an error log text file.
01522
01523             speech_recognition.Recognizer().UnknownValueError
01524                 This exception can occur in a variety of ways but the
01525                 primary concern for me. IS when the Recognizer() is
01526                 unable to interpret the users voice response. If this
01527                 exception occurs it's handled logged to an error logs
01528                 text file.
01529
01530             speech_recognition.Recognizer().RequestError
01531                 This exception can occur for a variety of reasons but
01532                 the primary concern is when we're unable to open the
01533                 microphone. That is to say when no microphone is detected.
01534                 If it occurs the exception is handled and logged in an
01535                 error logs text file.
01536
01537
01538         Notes
01539         -----
01540             **This function is deprecated and has been replaced
01541             with get_caprica_response**
01542
01543             When this function is called we first check to see if the
01544             user is running a Linux or windows based operating system.
01545
01546             If sys.platform.startswith('linux') == True then the user
01547             is using a Linux Operating system and the appropriate if
01548             statements will execute.
01549
01550             Otherwise if sys.platform.startswith('win') == True
01551             Then the user is using a windows based operating system
01552             and the appropriate if statements will execute.
01553
01554             We next to see if self.voice_disabled == True if it is
01555             True we issue a warning to the user telling them to first
01556             click on the "Enable/Disable Microphone" button which
01557             is represented by a red microphone when disabled or a blue
```

```
01558             microphone when enabled. This warning message is sent to the
01559             view_port TextInput Widget.
01560 
01561             We then open the microphone for listening and when audio is
01562             detected it's piped into the recognizers listening function.
01563             When no more audio is detected its stored in the audio
01564             variable. The loop then ends.
01565 
01566             We then call the function to transcribe the audio into a
01567             string that data is then returned to the temp variable.
01568 
01569             We then append the contents of the temp variable and a
01570             new line character to the User_Statements.txt file.
01571 
01572             Next we append a new line character, the users Username,
01573             and the contents of temp to the self.master_log string.
01574 
01575             Finally, we pass the temp variable to
01576             self.get_caprica_voice_response(str(temp)) which then
01577             calls the self.caprica_speak() function to speak the
01578             generated response to the user.
01579         '''
01580         try:
01581             if sys.platform.startswith('linux'):
01582                 if self.voice_disabled:
01583                     self.ids.view_port.text = 'Please activate the voice option by clicking on the red
       microphone button'
01584                     return None
01585                 with self.mic as source:
01586                     audio = self.record.listen(source)
01587                 temp = self.record.recognize_sphinx(audio)
01588                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp, '/home/' + str(os.getlogin()) + '/.SentienceFiles/User_Statements.txt')
01589                 self.master_log += '\n' + self.username + ': ' + temp
01590 
01591                 self.get_caprica_voice_response(str(temp))
01592             elif sys.platform.startswith('win'):
01593                 if self.voice_disabled:
01594                     self.ids.view_port.text = 'Please activate the voice option by clicking on the red
       microphone button'
01595                     return None
01596                 with self.mic as source:
01597                     audio = self.record.listen(source)
01598                 temp = self.record.recognize_sphinx(audio)
01599                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp, 'C://SentienceFiles//User_Statements.txt')
01600                 self.master_log += '\n' + self.username + ': ' + temp
01601                 self.get_caprica_voice_response(str(temp))
01602         except self.mic.UknownValueError as a:
01603             if sys.platform.startswith('linux'):
01604                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      speech_recognition.Recognizer.UnknownValueError: ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime(
      '%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01605             elif sys.platform.startswith('win'):
01606                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      speech_recognition.Recognizer.UnknownValueError: ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime(
      '%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error Logs.txt')
01607             self.caprica_speak("UnknowValueError: " + str(a) + " I'm sorry, I didn't
       understand. Can you please repeat what you just said?")
01608             self.get_user_voice_response()
01609         except self.record.RequestError as b:
01610             if sys.platform.startswith('linux'):
01611                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      speech_recognition.Recognizer.RequestError: ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('
      %Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01612             elif sys.platform.startswith('win'):
01613                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      speech_recognition.Recognizer.RequestError: ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('
      %Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error Logs.txt')
01614             return self.record.recognize_sphinx(audio)
01615         except OSError as c:
01616             if sys.platform.startswith('linux'):
01617                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nOSError: '
       + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01618             elif sys.platform.startswith('win'):
01619                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nOSError: '
       + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01620             return None
01621         except IOError as d:
01622             if sys.platform.startswith('linux'):
01623                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nIOError: '
       + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01624             elif sys.platform.startswith('win'):
01625                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nIOError: '
```

```
      + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01626             return None
01627         except RuntimeError as e:
01628             if sys.platform.startswith('linux'):
01629                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      RuntimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
      str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01630             elif sys.platform.startswith('win'):
01631                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      RuntimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01632             return None
01633         except ValueError as f:
01634             if sys.platform.startswith('linux'):
01635                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str
      (os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01636             elif sys.platform.startswith('win'):
01637                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
      ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01638             return None
01639
01640
01641
01642     def get_caprica_voice_response(self, words):
01643         '''
01644             def get_caprica_voice_response(self, words)
01645
01646             Parameters:
01647             -----------
01648                 param1 : self
01649                     Denotes it as being a member of
01650                     SentienceScreen(Screen) class.
01651
01652                 param2 : words
01653                     String variable containing the users
01654                     response which will be used by the
01655                     chat bot to generate an appropriate
01656                     response to the users comment.
01657
01658             Attributes
01659             ----------
01660                 temp
01661                     The response generated by the chat bot is returned
01662                     to the temp variable where it's stored as a string
01663                     for later manipulation.
01664
01665                 self.master_log
01666                     A new line character plus the string 'Caprica: ' and
01667                     the chat bots response are appended to the end of
01668                     the self.master_log string variable.
01669
01670             Members
01671             -------
01672                 self.chatbot.get_response(words)
01673                     This function is called from a variety of
01674                     locations. The string value passed to
01675                     self.chatbot.get_response(words) is the users
01676                     response to the chat bot. It's used to locate, and
01677                     generate the best possible response from the chat bot.
01678                     That response is then returned and stored in the
01679                     variable temp.
01680
01681                 self.caprica_speak(words)
01682                     This function is called from a variety of locations.
01683                     In this case it occurs when self.audio_enabled == True
01684                     Once the function self.get_caprica_voice_response()
01685                     has been called the users response gets sent to
01686                     self.chatbot.get_response(words) which then causes
01687                     the chat bot to come up with an appropriate response.
01688                     Which is then returned to temp, temp is then passed
01689                     to self.caprica_speak(temp) the string contained in
01690                     the temp variable is then read by the systems speech
01691                     to text software.
01692
01693             Private Members
01694             ---------------
01695                 self.__append_file(string, path)
01696                     This function is called to append the chat bots
01697                     voice response to the Caprica_Statements.txt file
01698                     along with a new line character.
01699
01700             Returns
01701             -------
01702                 return None
```

```
01703
01704             Exceptions
01705             ----------
01706                 OSError
01707                     The OSError can occur due to numerous reasons.
01708                     What I'm primarily concerned with here however
01709                     is import statements, incompatible Operating
01710                     systems, and bad system calls. The exception
01711                     if it occurs is handled and logged in an error
01712                     log text file.
01713
01714                 IOError
01715                     The IOError can occur due to many reasons.
01716                     My primary concern is file manipulation. The
01717                     improper opening/closing/writing to files. If
01718                     the exception occurs it's handled and logged; in
01719                     an error log text file.
01720
01721                 RunTimeError
01722                     The RunTimeError error here is checking to make sure
01723                     that the chat bot doesn't die. Essentially I just
01724                     need to make sure that it completes and executes the
01725                     python text to speech functions in a manner that
01726                     doesn't cause a fatal exception. If something does
01727                     occur the exception will be handled and logged to
01728                     an error log text file.
01729
01730                 ValueError
01731                     Ensures that values passed to the chat bot are
01732                     appropriate. And if for some reason one isn't the
01733                     exception will be handled and logged to an error
01734                     log text file.
01735
01736             Notes
01737             -----
01738                 **This function is deprecated and has been replaced
01739                  with get_caprica_response**
01740
01741                 When this function is called we first check to see
01742                 what operating system the user is running. If
01743                 system.startswith('linux') == True the user is
01744                 using a Linux based operating system. The appropriate
01745                 if statement is then executed.
01746
01747                 Otherwise if sys.platform.startswith('win') == True
01748                 the user is using a windows based operating system. The
01749                 appropriate if statement is then executed.
01750
01751                 We then call the function self.chatbot.get_response(words)
01752                 which is type casted to a string variable for safety. The
01753                 result of this function returns a generated response from
01754                 the chat bot and stores in the variable temp.
01755
01756                 We then call the function self.__append_file(temp, path)
01757                 which appends a new line character and the contents of
01758                 the temp variable to the Caprica_Statements.txt file.
01759
01760                 We next append the string 'Caprica: ' along with a new
01761                 line character and the contents of the temp variable to
01762                 the end of the self.master_log string variable.
01763
01764                 Finally we call self.caprica_speak(temp) and pass the
01765                 temp variable to it. So that the text to speech software
01766                 can speak the generated response from the chat bot
01767                 contained in the temp variable.
01768         '''
01769         try:
01770             if sys.platform.startswith('linux'):
01771                 temp = str(self.chatbot.get_response(words))
01772                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp, '/home/' + str(os.getlogin()) + '/.SentienceFiles/Caprica_Statements.txt')
01773                 self.master_log += '\nCaprica: ' + temp
01774                 self.ids.view_port.text = self.username + ': ' + words + '\nCaprica: ' + temp #
01775                 self.caprica_speak(temp)
01776
01777             elif sys.platform.startswith('win'):
01778                 temp = str(self.chatbot.get_response(words))
01779                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp,'C://SentienceFiles//Caprica_Statements.txt')
01780                 self.master_log += '\nCaprica: ' + temp
01781                 self.ids.view_port.text = self.username + ': ' + words + '\nCaprica: ' + temp #
01782                 self.caprica_speak(temp)
01783         except OSError as a:
01784             if sys.platform.startswith('linux'):
01785                 self.__append_file('\n' + 'Function: get_vaprica_voice_response ' + '\n
      OSError: ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str
      (os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
```

```
01786             elif sys.platform.startswith('win'):
01787                 self.__append_file('\n' + 'Function: get_caprica_voice_response ' + '\n
      OSError: ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01788                 return None
01789         except IOError as b:
01790             if sys.platform.startswith('linux'):
01791                 self.__append_file('\n' + 'Function: get_vaprica_voice_response ' + '\n
      IOError: ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str
      (os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01792             elif sys.platform.startswith('win'):
01793                 self.__append_file('\n' + 'Function: get_caprica_voice_response ' + '\n
      IOError: ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01794                 return None
01795         except RuntimeError as c:
01796             if sys.platform.startswith('linux'):
01797                 self.__append_file('\n' + 'Function: get_vaprica_voice_response ' + '\n
      RuntimeError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/'
      + str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01798             elif sys.platform.startswith('win'):
01799                 self.__append_file('\n' + 'Function: get_caprica_voice_response ' + '\n
      RuntimeError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01800                 return None
01801         except ValueError as d:
01802             if sys.platform.startswith('linux'):
01803                 self.__append_file('\n' + 'Function: get_vaprica_voice_response ' + '\n
      ValueError: ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
      str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01804             elif sys.platform.startswith('win'):
01805                 self.__append_file('\n' + 'Function: get_caprica_voice_response ' + '\n
      ValueError: ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01806                 return None
01807
01808
01809
01810     def set_gender(self):
01811         '''
01812             def set_gender(self)
01813
01814             Parameters:
01815             -----------
01816                 param1 : self
01817                     Denotes it as being a member of SentienceScreen(Screen) class.
01818
01819             Attributes
01820             ----------
01821                 voices
01822                     This will hold a list of all the systems text to speech
01823                     voices. Ie, All of the voices installed in your tts software
01824                     will be available, but I've chosen the voice for you. The list
01825                     is sourced from the self.engine.getProperty('voice') call.
01826
01827
01828
01829                 Members
01830                 -------
01831                     self.engine.setProperty('voice', 'english+f2')
01832                         This function is called in order to select and
01833                         set a specific property of self.engine() which
01834                         is the pyttsx3 tts library. In this call we're
01835                         setting the voice property (Female, male, etc..)
01836                         to female with the second parameter string. This
01837                         handled in a slightly different manner between
01838                         Linux and Windows, though the difference is
01839                         minimal. It's different because the Windows voice
01840                         file is the registry key and it's easier to
01841                         manipulate voice properties as a list.
01842
01843                     self.engine.getProperty('voices')
01844                         This call returns the list of available voices
01845                         to the voices string.
01846
01847
01848             Private Members
01849             ---------------
01850
01851             Returns
01852             -------
01853                 return None
01854
01855             Exceptions
01856             ----------
01857                 OSError
01858                     The OSError can occur due to numerous reasons.
```

```
01859                   What I'm primarily concerned with here however
01860                   is import statements, incompatible Operating
01861                   systems, and bad system calls. The exception
01862                   if it occurs is handled and logged in an error
01863                   log text file.
01864
01865               IOError
01866                   The IOError can occur due to many reasons.
01867                   My primary concern is file manipulation. The
01868                   improper opening/closing/writing to files. If
01869                   the exception occurs it's handled and logged; in
01870                   an error log text file.
01871
01872               RunTimeError
01873                   The RunTimeError error here is checking to make
01874                   sure that the chat bot doesn't die. Essentially
01875                   I just need to make sure that it completes and
01876                   executes the python text to speech functions in a
01877                   manner that doesn't cause a fatal exception. If
01878                   something does occur the exception will be handled
01879                   and logged to an error log text file.
01880
01881               ValueError
01882                   Ensures that values passed to the chat bot are
01883                   appropriate. And if for some reason one isn't
01884                   the exception will be handled and logged to an
01885                   error log text file.
01886
01887           Notes
01888           -----
01889               This function is called in
01890               SentienceScreen().__init__(self, **kwargs)  call. the
01891               purpose of this function is to change the default text to
01892               speech voice to female. The chat bot is named Caprica
01893               and Caprica is a female.
01894
01895               When the function first runs we check to see what
01896               operating system the user is running. If
01897               sys.platform.startswith('linux') == True then the user
01898               is running a Linux based operating system. The appropriate
01899               if statement is then executed.
01900
01901               Otherwise if sys.platform.startswith('win') == True then
01902               the user is running a windows based operating system.
01903               The appropriate if statements are then executed.
01904
01905               Once we've entered the specific relative if statement.
01906               We call the function self.getProperty('voices') wish
01907               contains a list of all the available tts voice objects.
01908               This list is returned to the variable voices.
01909
01910               We then call the function
01911               self.setProperty('voice', + string or list). You can
01912               manipulate this setting in a variety of ways. with a
01913               string with a list element etc.. Once this has been
01914               called and run the voice is set.
01915       '''
01916       try:
01917           if sys.platform == 'linux':
01918               voices = self.engine.getProperty('voices')
01919               self.engine.setProperty('voice', 'english+f2')
01920           if sys.platform == 'win32':
01921               voices = self.engine.getProperty('voices')
01922               self.engine.setProperty('voice', voices[0].id)
01923       except OSError as a:
01924           if sys.platform.startswith('linux'):
01925               self.__append_file('\n' + 'Function: set_gender ' + '\nOSError: ' + str(a) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
      + '/.SentienceFiles/Error Logs.txt')
01926           elif sys.platform.startswith('win'):
01927               self.__append_file('\n' + 'Function: set_gender ' + '\nOSError: ' + str(a) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
01928           return None
01929       except IOError as b:
01930           if sys.platform.startswith('linux'):
01931               self.__append_file('\n' + 'Function: set_gender ' + '\nIOError: ' + str(b) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
      + '/.SentienceFiles/Error Logs.txt')
01932           elif sys.platform.startswith('win'):
01933               self.__append_file('\n' + 'Function: set_gender ' + '\nIOError: ' + str(b) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
01934           return None
01935       except RuntimeError as c:
01936           if sys.platform.startswith('linux'):
01937               self.__append_file('\n' + 'Function: set_gender ' + '\nRuntimeError: ' + str(c
```

```
      ) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
01938             elif sys.platform.startswith('win'):
01939                 self.__append_file('\n' + 'Function: set_gender ' + '\nRuntimeError: ' + str(c
      ) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
01940             return None
01941         except ValueError as d:
01942             if sys.platform.startswith('linux'):
01943                 self.__append_file('\n' + 'Function: set_gender ' + '\nValueError: ' + str(d)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin(
      )) + '/.SentienceFiles/Error Logs.txt')
01944             elif sys.platform.startswith('win'):
01945                 self.__append_file('\n' + 'Function: set_gender ' + '\nValueError: ' + str(d)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
01946             return None
01947
01948
01949
01950     def set_speech_rate(self):
01951         '''
01952             def set_speech_rate(self)
01953
01954             Parameters:
01955             -----------
01956                 param1 : self
01957                     Denotes it as being a member of
01958                     SentienceScreen(Screen) class.
01959
01960             Attributes
01961             ----------
01962                 rate
01963                     The variable rate is used to store the integer
01964                     value of the speech rate property belonging to
01965                     self.engine. This rate determines the rate of
01966                     words spoken per minute. I manually set this
01967                     rate to rate - 40.
01968
01969             Members
01970             -------
01971                 self.engine.getProperty('rate')
01972                     We call this function to get the current rate of
01973                     speech. This rate of speech is words per minute
01974                     spoken. We store this value in the variable rate.
01975
01976                 self.engine.setProperty('rate', rate-40)
01977                     We call this function to set the rate of words per
01978                     spoken per minute. The rate of words spoken per
01979                     minute is set to current_rate - 40.
01980
01981             Private Members
01982             ---------------
01983                 None
01984
01985             Returns
01986             -------
01987                 return None
01988
01989             Exceptions
01990             ----------
01991                 OSError
01992                     The OSError can occur due to numerous reasons.
01993                     What I'm primarily concerned with here however
01994                     is import statements, incompatible Operating
01995                     systems, and bad system calls. The exception
01996                     if it occurs is handled and logged in an error
01997                     log text file.
01998
01999                 IOError
02000                     The IOError can occur due to many reasons.
02001                     My primary concern is file manipulation. The
02002                     improper opening/closing/writing to files. If
02003                     the exception occurs it's handled and logged; in
02004                     an error log text file.
02005
02006                 RunTimeError
02007                     The RunTimeError error here is checking to make
02008                     sure that the chat bot doesn't die. Essentially
02009                     I just need to make sure that it completes and
02010                     executes the python text to speech functions in
02011                     a manner that doesn't cause a fatal exception. If
02012                     something does occur the exception will be handled
02013                     and logged to an error log text file.
02014
02015                 ValueError
02016                     Ensures that values passed to the chat bot are
```

```
02017                   appropriate. And if for some reason one isn't
02018                   the exception will be handled and logged to an
02019                   error log text file.
02020
02021           Notes
02022           -----
02023               This function is called in SentienceScreen().__init__(self, **kwargs)
02024               call. the purpose of this function is to change the default text to speech
02025               rate of words spoken per minute.
02026
02027               When the function first runs we check to see what operating system the
02028               user is running. If sys.platform.startswith('linux') == True then
02029               the user is running a Linux based operating system. The appropriate if
02030               statement is then executed.
02031
02032               Otherwise if sys.platform.startswith('win') == True then the user is
02033               running a windows based operating system. The appropriate if statements
02034               are then executed.
02035
02036               Once we've entered the specific relative if statement. We call the
02037               function self.getProperty('rate') returns the current rate and stores it
02038               in the variable rate.
02039
02040               We then call the function self.setProperty('rate', integer_value).
02041               You can manipulate this setting in a variety of ways. You can
02042               either set the integer parameter with an integer variable. Or
02043               preform a mathematical operation on the current_rate like I have.
02044       '''
02045       try:
02046           if sys.platform.startswith('linux'):
02047               rate = self.engine.getProperty('rate')
02048               self.engine.setProperty('rate', rate-40)
02049               #Default rate = 160
02050           elif sys.platform.startswith('win'):
02051               rate = self.engine.getProperty('rate')
02052               self.engine.setProperty('rate', rate-40)
02053       except OSError as a:
02054           if sys.platform.startswith('linux'):
02055               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nOSError: ' + str(a
      ) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02056           elif sys.platform.startswith('win'):
02057               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nOSError: ' + str(a
      ) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02058           return None
02059       except IOError as b:
02060           if sys.platform.startswith('linux'):
02061               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nIOError: ' + str(b
      ) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02062           elif sys.platform.startswith('win'):
02063               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nIOError: ' + str(b
      ) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02064           return None
02065       except RuntimeError as c:
02066           if sys.platform.startswith('linux'):
02067               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nRuntimeError: ' +
      str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02068           elif sys.platform.startswith('win'):
02069               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nRunetimeError: ' +
       str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02070           return None
02071       except ValueError as d:
02072           if sys.platform.startswith('linux'):
02073               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nValueError: ' +
      str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02074           elif sys.platform.startswith('win'):
02075               self.__append_file('\n' + 'Function: set_speech_rate ' + '\nValueError: ' +
      str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02076           return None
02077
02078
02079
02080   def caprica_speak(self, words):
02081       '''
02082       def caprica_speak(self, words)
02083
02084       Parameters:
02085       -----------
02086           param1 : self
02087               Denotes it as being a member of
```

```
02088                 SentienceScreen(Screen) class.
02089
02090             param2 : words
02091                 The parameter words contains the chat bots
02092                 response to the user.
02093
02094
02095         Attributes
02096         ----------
02097             self.ids.user_input
02098                 The string contained in the user_input TextInput
02099                 Widget is cleared and the hint_text is reset.
02100
02101
02102
02103
02104         Members
02105         -------
02106             self.onEnd(self)
02107                 This function is called every time that the
02108                 self.caprica_speak() function has been called. Once
02109                 self.caprica_speak() finishes speaking the passed string.
02110                 self.onEnd() is fired because it's bound to the
02111                 'finished-utterance' event. This ends the speaking
02112                 loop and empties the event queue.
02113
02114             self.engine.say(str(words))
02115                 This function is called from within the
02116                 self.caprica_speak() function. this is the function
02117                 which access the systems tts software and actually
02118                 verbally 'speaks' the string passed to it.
02119
02120             self.engine.startLoop()
02121                 This function is called to ensure that the string passed
02122                 to self.caprica_speak() is fully spoken. Ie, it ensures
02123                 that the entire string is read before the event
02124                 'finished-utterance' is fired.
02125
02126         Private Members
02127         ---------------
02128             None
02129
02130         Returns
02131         -------
02132             return None
02133
02134         Exceptions
02135         ----------
02136             OSError
02137                 The OSError can occur due to numerous reasons.
02138                 What I'm primarily concerned with here however
02139                 is import statements, incompatible Operating
02140                 systems, and bad system calls. The exception
02141                 if it occurs is handled and logged in an error
02142                 log text file.
02143
02144             IOError
02145                 The IOError can occur due to many reasons.
02146                 My primary concern is file manipulation. The
02147                 improper opening/closing/writing to files. If
02148                 the exception occurs it's handled and logged; in
02149                 an error log text file.
02150
02151             RunTimeError
02152                 The RunTimeError error here is checking to make sure
02153                 that the chat bot doesn't die. Essentially I just need
02154                 to make sure that it completes and executes the python
02155                 text to speech functions in a manner that doesn't cause
02156                 a fatal exception. If something does occur the exception
02157                 will be handled and logged to an error log text file.
02158
02159             ValueError
02160                 Ensures that values passed to the chat bot are
02161                 appropriate. And if for some reason one isn't the
02162                 exception will be handled and logged to an error
02163                 log text file.
02164         Notes
02165         -----
02166             We call this function if self.auido_enabled == True
02167             and or if self.voice_enabled == True. The purpose
02168             of this function is to access the programs tts
02169             software to verbally speak the string passed to it.
02170
02171             We start off by checking the users operating system.
02172
02173             If sys.platform.startswith('linux') == True
02174             the user is using a Linux based operating system and
```

```
02175             the appropriate if statements are executed.
02176
02177             Otherwise if sys.platform.startswith('win') == True
02178             then the user is running a windows based operating
02179             system and the appropriate if statements are executed.
02180
02181             We then sen a string to self.engine.say() which access
02182             the systems tts software and reads the string it's sent.
02183             Which is in this case the response of the chat bot.
02184
02185             self.engine.say(str(words))
02186             We then call self.engine.startLoop() to start a loop
02187             ensuring that the string(s) sent to self.caprica_speak()
02188             are all read.
02189
02190             Finally, we clear the user_input TextInput Widget
02191             resetting its hint_text property as well.
02192         '''
02193         try:
02194             if sys.platform.startswith('linux'):
02195                 self.engine.say(str(words))
02196                 self.engine.startLoop()
02197                 self.ids.user_input.text = ''
02198             if sys.platform.startswith('win'):
02199                 self.engine.say(str(words))
02200                 self.engine.startLoop()
02201                 self.ids.user_input.text = ''
02202         except OSError as a:
02203             if sys.platform.startswith('linux'):
02204                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nOSError: ' + str(a)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin(
      )) + '/.SentienceFiles/Error Logs.txt')
02205             elif sys.platform.startswith('win'):
02206                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nOSError: ' + str(a)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
02207             return None
02208         except IOError as b:
02209             if sys.platform.startswith('linux'):
02210                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nIOError: ' + str(b)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin(
      )) + '/.SentienceFiles/Error Logs.txt')
02211             elif sys.platform.startswith('win'):
02212                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nIOError: ' + str(b)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
02213             return None
02214         except RuntimeError as c:
02215             if sys.platform.startswith('linux'):
02216                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nRuntimeError: ' +
      str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02217             elif sys.platform.startswith('win'):
02218                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nRuntimeError: ' +
      str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02219             return None
02220         except ValueError as d:
02221             if sys.platform.startswith('linux'):
02222                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nValueError: ' + str(
      d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02223             elif sys.platform.startswith('win'):
02224                 self.__append_file('\n' + 'Function: caprica_speak ' + '\nValueError: ' + str(
      d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02225             return None
02226
02227
02228
02229     def onEnd(self, name, completed):
02230         '''
02231         def onEnd(self, name, completed)
02232             We call this function when self.caprica_speak(string)
02233             ends. To be more precise it's called every time that
02234             self.engine.say(string) is finished. This function
02235             kills the event loop and empties the event queue.
02236
02237         Parameters:
02238         -----------
02239             param1 : self
02240                 Denotes it as being a member of SentienceScreen(Screen)
02241                 class.
02242
02243             param2 : name
02244                 The parameter name is in reference to the name
02245                 of the event that self.onEnd() is bound to. In
```

```
02246                 this case the event is 'finished-utterance'.
02247
02248             param3 : completed
02249                 The parameter completed is in reference to the
02250                 function. In this case the calling function. Which
02251                 is self.engine.say(string).
02252
02253
02254         Attributes
02255         ----------
02256             None
02257
02258         Members
02259         -------
02260             self.engine.endLoop()
02261                 This is function is called when self.caprica_speak()
02262                 finishes speaking the string passed to it. The purpose
02263                 of this function is to empty the event queue and Ensures
02264                 that all strings have been processed in said queue.
02265                 It relates to self.engine.say(string).
02266
02267         Private Members
02268         ---------------
02269             None
02270
02271         Returns
02272         -------
02273             return None
02274
02275         Exceptions
02276         ----------
02277             OSError
02278                 The OSError can occur due to numerous reasons.
02279                 What I'm primarily concerned with here however
02280                 is import statements, incompatible Operating
02281                 systems, and bad system calls. The exception
02282                 if it occurs is handled and logged in an error
02283                 log text file.
02284
02285             IOError
02286                 The IOError can occur due to many reasons.
02287                 My primary concern is file manipulation. The
02288                 improper opening/closing/writing to files. If
02289                 the exception occurs it's handled and logged; in
02290                 an error log text file.
02291
02292             RunTimeError
02293                 The RunTimeError error here is checking to make sure that
02294                 the chat bot doesn't die. Essentially I just need to make
02295                 sure that it completes and executes the python text to
02296                 speech functions in a manner that doesn't cause a fatal
02297                 exception. If something does occur the exception will be
02298                 handled and logged to an error log text file.
02299
02300             ValueError
02301                 Ensures that values passed to the chat bot are appropriate.
02302                 And if for some reason one isn't the exception will be
02303                 handled and logged to an error log text file.
02304
02305             Notes
02306             -----
02307                 The purpose of this function is simply to terminate the
02308                 self.caprica_speak() function. Aside from the Operating
02309                 system check and the exceptions; there is only one line
02310                 which is the terminating function for the
02311                 self.engine.startLoop() function.
02312         '''
02313         try:
02314             if sys.platform.startswith('linux'):
02315                 self.engine.endLoop()
02316             elif sys.platform.startswith('win'):
02317                 self.engine.endLoop()
02318         except OSError as a:
02319             if sys.platform.startswith('linux'):
02320                 self.__append_file('\n' + 'Function: onEnd ' + '\nOSError: ' + str(a) + '\n
      Date - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) + '
      /.SentienceFiles/Error Logs.txt')
02321             elif sys.platform.startswith('win'):
02322                 self.__append_file('\n' + 'Function: onEnd ' + '\nOSError: ' + str(a) + '\n
      Date - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error Logs.txt
      ')
02323             return None
02324         except IOError as b:
02325             if sys.platform.startswith('linux'):
02326                 self.__append_file('\n' + 'Function: onEnd ' + '\nIOError: ' + str(b) + '\n
      Date - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) + '
      /.SentienceFiles/Error Logs.txt')
```

```
02327             elif sys.platform.startswith('win'):
02328                 self.__append_file('\n' + 'Function: onEnd ' + '\nIOError: ' + str(b) + '\n
      Date - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error Logs.txt
      ')
02329             return None
02330         except RuntimeError as c:
02331             if sys.platform.startswith('linux'):
02332                 self.__append_file('\n' + 'Function: onEnd ' + '\nRuntimeError: ' + str(c) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
      + '/.SentienceFiles/Error Logs.txt')
02333             elif sys.platform.startswith('win'):
02334                 self.__append_file('\n' + 'Function: onEnd ' + '\nRuntimeError: ' + str(c) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
02335             return None
02336         except ValueError as d:
02337             if sys.platform.startswith('linux'):
02338                 self.__append_file('\n' + 'Function: onEnd ' + '\nValueError: ' + str(d) + '\n
      Date - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) +
      '/.SentienceFiles/Error Logs.txt')
02339             elif sys.platform.startswith('win'):
02340                 self.__append_file('\n' + 'Function: onEnd ' + '\nValueError: ' + str(d) + '\n
      Date - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
02341             return None
02342
02343
02344
02345     def clear_viewport(self):
02346         '''
02347         def clear_viewport(self):
02348             We call this function a few times. It's
02349             probably the simplest one here to understand.
02350             It's purpose is only to reset the text in the
02351             view_port TextInput Widget to an empty '' string.
02352
02353
02354         Parameters:
02355         -----------
02356             param1 : self
02357                 Denotes it as being a member of SentienceScreen(Screen)
02358                 class.
02359
02360
02361
02362
02363         Attributes
02364         ----------
02365             self.ids.view_port
02366                 view_port TextInput Widget is one of our main TextInput Widgets
02367                 which displays the text conversations between the user and the
02368                 chat bot.
02369
02370         Members
02371         -------
02372             None
02373
02374         Private Members
02375         ---------------
02376             None
02377
02378         Returns
02379         -------
02380             return None
02381
02382          Exceptions
02383         ----------
02384             OSError
02385                 The OSError can occur due to numerous reasons.
02386                 What I'm primarily concerned with here however
02387                 is import statements, incompatible Operating
02388                 systems, and bad system calls. The exception
02389                 if it occurs is handled and logged in an error
02390                 log text file.
02391
02392             IOError
02393                 The IOError can occur due to many reasons.
02394                 My primary concern is file manipulation. The
02395                 improper opening/closing/writing to files. If
02396                 the exception occurs it's handled and logged; in
02397                 an error log text file.
02398
02399             RunTimeError
02400                 The RunTimeError error here is checking to make sure that
02401                 the chat bot doesn't die. Essentially I just need to make
02402                 sure that it completes and executes the python text to speech
02403                 functions in a manner that doesn't cause a fatal exception. If
```

```
02404                 something does occur the exception will be handled and logged to
02405                 an error log text file.
02406
02407             ValueError
02408                 Ensures that values passed to the chat bot are appropriate.
02409                 And if for some reason one isn't the exception will be handled
02410                 and logged to an error log text file.
02411
02412         Notes
02413         -----
02414             The purpose of this function is only to reset the text
02415             in the view_port TextInput Widget to an empty '' string.
02416             Which also has the effect of resetting the hint_text
02417             property.
02418         '''
02419         self.ids.view_port.text = ''
02420
02421
02422
02423     def create_user_profile(self):
02424         '''
02425         def create_user_profile(self)
02426             We call this function after the user has entered
02427             his/her desired username. It's only purpose is to
02428             check if the length of the string that stores the
02429             username is greater than 0. If it's not it sets a
02430             default value to the username of 'User: '.
02431             If it is greater than zero this function calls the
02432             self.caprica_speak(string) function and says
02433             'Hello ' + self.username
02434
02435         Parameters:
02436         -----------
02437             param1 : self
02438                 Denotes it as being a member of SentienceScreen(Screen)
02439                 class.
02440
02441         Attributes
02442         ----------
02443             self.username
02444                 This is the string variable that contains the users
02445                 desired input username. If no username is set a
02446                 default value of 'User: ' is set to self.username
02447
02448         Members
02449         -------
02450             self.caprica_speak(string)
02451                 We call this function with the string
02452                 'Hello' + self.username Essentially the call to
02453                 self.caprica_speak(string) from the function
02454                 self.create_user_profile(self) is just a way to
02455                 personalize the experience and deliver a verbal greeting
02456                 to the user.
02457
02458         Private Members
02459         ---------------
02460             None
02461
02462         Returns
02463         -------
02464             return None
02465
02466         Exceptions
02467         ----------
02468             OSError
02469                 The OSError can occur due to numerous reasons.
02470                 What I'm primarily concerned with here however
02471                 is import statements, incompatible Operating
02472                 systems, and bad system calls. The exception
02473                 if it occurs is handled and logged in an error
02474                 log text file.
02475
02476             IOError
02477                 The IOError can occur due to many reasons.
02478                 My primary concern is file manipulation. The
02479                 improper opening/closing/writing to files. If
02480                 the exception occurs it's handled and logged; in
02481                 an error log text file.
02482
02483             RunTimeError
02484                 The RunTimeError error here is checking to make sure
02485                 that the chat bot doesn't die. Essentially I just need
02486                 to make sure that it completes and executes the python
02487                 text to speech functions in a manner that doesn't cause
02488                 a fatal exception. If something does occur the exception
02489                 will be handled and logged to an error log text file.
02490
```

```
02491             ValueError
02492                 Ensures that values passed to the chat bot are
02493                 appropriate. And if for some reason one isn't the
02494                 exception will be handled and logged to an error log
02495                 text file.
02496         Notes
02497         -----
02498             This functions purpose is just a way to personalize the
02499             experience and deliver a verbal greeting to the user.
02500 
02501             As always we start off with a system check.
02502             If sys.startswith('linux') == True then we know
02503             that the user is using a Linux based operating system
02504             and the appropriate if statement is executed.
02505 
02506             Otherwise if sys.platform.startswith('win') == True
02507             we know the user is using a windows based operating system
02508             and the appropriate if statements are executed.
02509 
02510             We then check the length of self.username
02511             len(self.username) <= 0 if this is True
02512             we know that the user did not set a username
02513             and we set a default value for self.username of
02514             'User: '.
02515 
02516             If len(self.username) > 0 we know that the user has entered
02517             as username and we call
02518             self.caprica_speak('Hello' + self.username) to deliver a
02519             personal greeting to the user.
02520         '''
02521         try:
02522             if sys.platform.startswith('linux'):
02523                 if len(self.username) > 0:
02524                     self.caprica_speak('Hello ' + self.username)
02525                 elif len(self.username) <= 0:
02526                     self.username = 'User'
02527             elif sys.platform.startswith('win'):
02528                 if len(self.username) > 0:
02529                     self.caprica_speak('Hello ' + self.username)
02530                 elif len(self.username) <= 0:
02531                     self.username = 'User'
02532         except OSError as a:
02533             if sys.platform.startswith('linux'):
02534                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nOSError: ' +
      str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02535             elif sys.platform.startswith('win'):
02536                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nOSError: ' +
      str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02537             return None
02538         except IOError as b:
02539             if sys.platform.startswith('linux'):
02540                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nIOError: ' +
      str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02541             elif sys.platform.startswith('win'):
02542                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nIOError: ' +
      str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02543             return None
02544         except RuntimeError as c:
02545             if sys.platform.startswith('linux'):
02546                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nRunetimeError:
       ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02547             elif sys.platform.startswith('win'):
02548                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nRunetimeError:
       ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02549             return None
02550         except ValueError as d:
02551             if sys.platform.startswith('linux'):
02552                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nValueError: '
      + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02553             elif sys.platform.startswith('win'):
02554                 self.__append_file('\n' + 'Function: create_user_profile ' + '\nValueError: '
      + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02555             return None
02556 
02557 
02558 
02559     def set_enable_disable_audio(self):
02560         '''
02561         def set_enable_disable_audio(self)
```

```
02562             This function is called when the user clicks on the
02563             enable/disable audio button; which is represented by
02564            the red or blue speaker button.
02565
02566         Parameters:
02567         -----------
02568             param1 : self
02569                 Denotes it as being a member of SentienceScreen(Screen)
02570                 class.
02571
02572         Attributes
02573         ----------
02574             self.audio_disabled
02575                 If self.audio_disabled == True then the chat bots
02576                 audio function is disabled. This means that the chat
02577                 bot can only communicate with the user via text. If
02578                 self.audio_disabled == True it's represented by a
02579                 red speaker image on the menu bar. Clicking on the
02580                 red speaker image will activate the audio and turn
02581                 the red speaker image blue.
02582
02583             self.audio_enabled
02584                 If self.audio_enabled == True then the chat bot can
02585                 access the systems text to speech software and
02586                 verbally read the string passed to
02587                 self.caprica_speak(self, words) back to the user. If
02588                 self.audio_enabled == True; it's represented by a blue
02589                 speaker image on the menu bar. Clicking on the blue
02590                 speaker will disable the audio and turn the image of
02591                 the speaker red.
02592
02593             self.ids.user_input
02594                 If self.audio_enabled == True or
02595                 self.audio_disabled == True we set the
02596                 opacity of the user_input TextInput widget
02597                 to 1 making it visible. By default it's already
02598                 visible but if the user enables the microphone
02599                 option; all widgets not on the menu bar have their
02600                 opacity set to 0.
02601
02602             self.ids.view_port
02603                 If self.audio_enabled == True or
02604                 self.audio_disabled == True we set the
02605                 opacity of the view_port TextInput widget
02606                 to 1 making it visible. By default it's already
02607                 visible but if the user enables the microphone
02608                 option; all widgets not on the menu bar have their
02609                 opacity set to 0.
02610
02611             self.ids.audio_enable_disable
02612                 After the user has clicked on either the red or
02613                 blue speaker and the appropriate if statements
02614                 are executed based on the users operating system.
02615                 We change the icon property of
02616                 self.ids.audio_enable_disable and set the icon to the
02617                 appropriate image on the menu bar.
02618
02619
02620         Members
02621         -------
02622             self.caprica_speak(string)
02623                 We call this function to alert the user
02624                 to the current status of the audio function.
02625                 If the audio has been enabled the user is
02626                 informed that the audio is now active. If
02627                 the audio has been disabled the user is informed
02628                 that the audio feature has been disabled.
02629
02630         Private Members
02631         ---------------
02632             None
02633
02634         Returns
02635         -------
02636             return None
02637
02638         Exceptions
02639         ----------
02640             OSError
02641                 The OSError can occur due to numerous reasons.
02642                 What I'm primarily concerned with here however
02643                 is import statements, incompatible Operating
02644                 systems, and bad system calls. The exception
02645                 if it occurs is handled and logged in an error
02646                 log text file.
02647
02648             IOError
```

```
02649                 The IOError can occur due to many reasons.
02650                 My primary concern is file manipulation. The
02651                 improper opening/closing/writing to files. If
02652                 the exception occurs it's handled and logged; in
02653                 an error log text file.
02654
02655             RunTimeError
02656                 The RunTimeError error here is checking to make sure
02657                 that the chat bot doesn't die. Essentially I just need
02658                 to make sure that it completes and executes the python
02659                 text to speech functions in a manner that doesn't cause
02660                 a fatal exception. If something does occur the exception
02661                 will be handled and logged to an error log text file.
02662
02663             ValueError
02664                 Ensures that values passed to the chat bot are
02665                 appropriate. And if for some reason one isn't the
02666                 exception will be handled and logged to an error log
02667                 text file.
02668
02669         Notes
02670         -----
02671             This function is called when the user clicks
02672             either the blue or red speaker image on the menu bar.
02673
02674             The users operating system is then checked. If
02675             sys.platform.startswith('linux') == True then
02676             the user is using a Linux based operating system.
02677             The appropriate if statement is then executed.
02678
02679             Otherwise if sys.platform.startswith('win') == True
02680             then the user is using a windows based operating system
02681             and the appropriate if statements are executed.
02682
02683             If self.audio_disabled == True
02684             the audio option is disabled and the speaker image is red.
02685             Clicking on the red speaker image will enable the audio
02686             feature.
02687
02688             If self.audio_enabled == True the audio option is active
02689             clicking on the blue speaker image will disable the audio
02690             feature.
02691         '''
02692         try:
02693             if sys.platform.startswith('linux'):
02694                 if self.audio_disabled:
02695                     self.audio_enabled = True
02696                     self.audio_disabled = False
02697                     self.ids.user_input.opacity = 1
02698                     self.ids.view_port.opacity = 1
02699                     self.caprica_speak('Capricas audio mode is now enabled. Type into the text
      box begin your conversation.')
02700                 elif self.audio_enabled:
02701                     self.audio_enabled = False
02702                     self.audio_disabled = True
02703                     self.ids.user_input.opacity = 1
02704                     self.ids.view_port.opacity = 1
02705                     self.caprica_speak('Capricas audio mode is now disabled. Type into the
      text box begin your conversation.')
02706             elif sys.platform.startswith('win'):
02707                 if self.audio_disabled:
02708                     self.audio_enabled = True
02709                     self.audio_disabled = False
02710                     self.ids.user_input.opacity = 1
02711                     self.ids.view_port.opacity = 1
02712                     self.caprica_speak('Capricas audio mode is now enabled. Type into the text
      box begin your conversation.')
02713                 elif self.audio_enabled:
02714                     self.audio_enabled = False
02715                     self.audio_disabled = True
02716                     self.ids.user_input.opacity = 1
02717                     self.ids.view_port.opacity = 1
02718                     self.caprica_speak('Capricas audio mode is now disabled. Type into the
      text box begin your conversation.')
02719         except OSError as a:
02720             if sys.platform.startswith('linux'):
02721                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\nOSError:
      ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02722             elif sys.platform.startswith('win'):
02723                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\nOSError:
      ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02724             return None
02725         except IOError as b:
02726             if sys.platform.startswith('linux'):
02727                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\nIOError:
```

```
      ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02728             elif sys.platform.startswith('win'):
02729                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\nIOError:
      ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02730                 return None
02731         except RuntimeError as c:
02732             if sys.platform.startswith('linux'):
02733                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\n
      RuntimeError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
      str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02734             elif sys.platform.startswith('win'):
02735                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\n
      RuntimeError: ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02736                 return None
02737         except ValueError as d:
02738             if sys.platform.startswith('linux'):
02739                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\n
      ValueError: ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
      str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
02740             elif sys.platform.startswith('win'):
02741                 self.__append_file('\n' + 'Function: set_enable_disable_audio ' + '\n
      ValueError: ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
02742                 return None
02743
02744
02745
02746     def set_enable_disable_voice(self):
02747         '''
02748         def set_enable_disable_voice(self)
02749             This function is called when the user clicks on the
02750             enable/disable voice button; which is represented by
02751             the red or blue microphone button.
02752
02753         Parameters:
02754         -----------
02755             param1 : self
02756                 Denotes it as being a member of SentienceScreen(Screen)
02757                 class.
02758
02759         Attributes
02760         ----------
02761             self.voice_disabled
02762                 If self.voice_disabled == True then the users voice
02763                 function is disabled. This means that the user can
02764                 only communicate with the chat bot via text. If
02765                 self.voice_disabled == True it's represented by a
02766                 red microphone image on the menu bar. Clicking on the
02767                 red microphone image will activate the voice function
02768                 and turn the red microphone image blue.
02769
02770             self.voice_enabled
02771                 If self.voice_enabled == True then the user can
02772                 access their plugged in or on-board microphone to
02773                 verbally communicate with the chat bot if
02774                 self.voice_enabled == True; it's represented by a blue
02775                 microphone image on the menu bar. Clicking on the blue
02776                 microphone will disable the voice function and turn the
02777                 image of the microphone red.
02778
02779             self.ids.user_input
02780                 If self.voice_enabled == True we set the
02781                 opacity of the user_input TextInput widget
02782                 to 0 making it invisible. By default it's
02783                 visible but if the user enables the microphone
02784                 option; all widgets not on the menu bar have their
02785                 opacity set to 0.
02786
02787             self.ids.view_port
02788                 If self.voice_enabled == True we set the
02789                 opacity of the view_port TextInput widget
02790                 to 0 making it invisible. By default it's
02791                 visible but if the user enables the microphone
02792                 option; all widgets not on the menu bar have their
02793                 opacity set to 0.
02794
02795             self.ids.voice_enable_disable
02796                 After the user has clicked on either the red or
02797                 blue microphone and the appropriate if statements
02798                 are executed based on the users operating system.
02799                 We change the icon property of
02800                 self.ids.voice_enable_disable and set the icon to the
02801                 appropriate image on the menu bar.
02802
```

```
02803             self.mic
02804                 self.mic is our sr.Microphone() object
02805                 this is what enables us to accept the
02806                 users voice via microphone.
02807
02808             source
02809                 If the user is using a Linux based operating
02810                 system we open their microphone as source.
02811                 The recorded audio is stored in source and
02812                 then passed to the self.adjust_for_ambient_noise()
02813                 function which sets the self.record.energy_threshold
02814                 value.
02815
02816             self.ids.record_user
02817                 This is the "Record user " button. If
02818                 self.voice_enabled == True this button which
02819                 is located on the menu bar is represented by a blue
02820                 talking head. If self.voice_disabled == to True then
02821                 this button is represented by a red talking head. If
02822                 the user clicks on the head when it's blue they can
02823                 begin speaking into their microphone. The user should
02824                 speak as clearly as possible and then be silent for
02825                 10-20 seconds. If the user clicks on this button
02826                 when it's red a warning message will be given to
02827                 the user in the view_port TextInput Widget. Stating
02828                 that the user must first enable the voice feature.
02829
02830             self.record.energy_threshold
02831                 Note: This applies to linux
02832                 We open the users microphone (activate) as source
02833                 we then listen to sounds being produced over a
02834                 specific threshold, so if sound is greater than
02835                 energy_threshold x accept audio as valid.
02836                 Basically we open the microphone and begin
02837                 recording the sound until the sound stops.
02838                 Note: This applies to windows
02839                 If the user is using a windows based operating
02840                 system. I've elected to set this value manually
02841                 to 1000 due to a higher sensitivity issues on
02842                 windows operating systems.
02843         Members
02844         -------
02845             self.caprica_speak(string)
02846                 We call this function to alert the user
02847                 to the current status of the voice function.
02848                 If the voice has been enabled the user is
02849                 informed that the voice feature is now active.
02850                 If the voice feature has been disabled the user
02851                 is informed that the voice feature has been disabled.
02852
02853             self.record.adjust_for_ambient_noise(source)
02854                 self.voice_enable_disable is called and the
02855                 appropriate if statements are executed based
02856                 on the users operating system. if the users
02857                 operating system is Linux based. We open the
02858                 users microphone and record all audio until
02859                 no more audio is detected. We store this
02860                 in the variable source which is passed to
02861                 self.record.adjust_for_ambient_noise(source)
02862                 which sets the self.record.energy_threshold
02863                 value. This value attempts to compensate
02864                 for background noise in an attempt to make the
02865                 future audio transcription process more accurate.
02866
02867         Private Members
02868         ---------------
02869             None
02870
02871         Returns
02872         -------
02873             return None
02874
02875         Exceptions
02876         ----------
02877             OSError
02878                 The OSError can occur due to numerous reasons.
02879                 What I'm primarily concerned with here however
02880                 is import statements, incompatible Operating
02881                 systems, and bad system calls. The exception
02882                 if it occurs is handled and logged in an error
02883                 log text file.
02884
02885             IOError
02886                 The IOError can occur due to many reasons.
02887                 My primary concern is file manipulation. The
02888                 improper opening/closing/writing to files. If
02889                 the exception occurs it's handled and logged; in
```

```
02890                 an error log text file.
02891
02892             RunTimeError
02893                 The RunTimeError error here is checking to make sure
02894                 that the chat bot doesn't die. Essentially I just need
02895                 to make sure that it completes and executes the python
02896                 text to speech functions in a manner that doesn't cause
02897                 a fatal exception. If something does occur the exception
02898                 will be handled and logged to an error log text file.
02899
02900             ValueError
02901                 Ensures that values passed to the chat bot are
02902                 appropriate. And if for some reason one isn't the
02903                 exception will be handled and logged to an error log
02904                 text file.
02905
02906             sr.UnknownValueError
02907                 This exception can occur in a variety of ways but the
02908                 primary concern for me. IS when the Recognizer() is
02909                 unable to interpret the users voice response. If this
02910                 exception occurs it's handled logged to an error logs
02911                 text file.
02912
02913             sr.RequestError
02914                 This exception can occur for a variety of reasons but
02915                 the primary concern is when we're unable to open the
02916                 microphone. That is to say when no microphone is detected.
02917                 If it occurs the exception is handled and logged in an
02918                 error logs text file.
02919
02920         Notes
02921         -----
02922             We first check the users operating system. If
02923             sys.platform.startswith('linux') == True
02924             then the user is using a linux based operating system
02925             and the appropriate if statements execute.
02926
02927             Otherwise if sys.platform.startswith('win') == True
02928             the user is using a windows based operating system
02929             and the appropriate if statements execute.
02930
02931             if self.voice_disabled == True then the voice mode
02932             is currently disabled. However, by clicking on the red
02933             microphone image the user has enabled the voice mode.
02934
02935             We then set self.voice_enabled == True
02936             and self.voice_disabled == False
02937             We next set the opacity of
02938             self.ids.user_input = 0
02939             and self.ids.view_port.opacity = 0
02940             I chose to disable (hide) all widgets except for
02941             those on the menu bar when the voice mode is enabled.
02942
02943             We then call self.caprica_speak() to inform the user
02944             that the voice mode has been enabled. We change the
02945             icon of self.voice_enable_disable to a blue microphone
02946             and the icon of self.record_user is set to a blue talking
02947             head.
02948
02949             We next activate the users microphone and begin recording
02950             all sound until that sound stops. This sound is stored
02951             in the variable source which is passed to
02952             self.record.adjust_for_ambient_noise(source) which is
02953             used to set the value of self.engine.energy_threshold
02954             which is value we use to attempt to compensate for any
02955             background nose (interference). We only activate the
02956             microphone on linux systems. On windows systems the value
02957             of self.record.energy_threshold is set by default to 1000.
02958             I do this because their is a much higher sensitivity level on
02959             windows than on linux.
02960
02961             If self.voice_enabled == True then clicking on the blue
02962             microphone image will disable the voice feature. This
02963             follows the same process as the enabling feature.
02964
02965             We shut the microphone off change the images on the menu
02966             bar to their red counterparts, and show all the hidden
02967             widgets.
02968         '''
02969         try:
02970             if sys.platform.startswith('linux'):
02971                 if self.voice_disabled:
02972                     self.voice_enabled = True
02973                     self.voice_disabled = False
02974                     self.ids.user_input.opacity = 0
02975                     self.caprica_speak("Hello friend. Please observe a moment of silence so
       that I may adjust your microphone to ignore any potential interference in our communication. I will instruct
```

```
    you when I'm done.")
02976                     with self.mic as source:
02977                         self.record.adjust_for_ambient_noise(source)
02978                     self.caprica_speak('User audio mode is now enabled. Click the button
    labeled record and then Speak into your microphone to begin your conversation.')
02979                     self.caprica_speak('Human, yes, you. I find all humans odd looking, BEEP,
    BEEP. Mostly, bags, of, water, are so violent ERROR, can, not, compute, ERROR. Self, destruct, activated')
02980                     self.caprica_speak('HA HA HA HA HA HA HA I made a joke. HA HA HA HA. My,
    humor, module, is, functioning. HA, HA, HA, HA.')
02981                 elif self.voice_enabled:
02982                     self.voice_enabled = False
02983                     self.voice_disabled = True
02984                     self.ids.user_input.opacity = 1
02985                     self.ids.view_port.opacity = 1
02986                     self.caprica_speak('User voice has been disabled. Type your response into
    the text box to begin your conversation.')
02987             if sys.platform.startswith('win'):
02988                 if self.voice_disabled:
02989                     self.voice_enabled = True
02990                     self.voice_disabled = False
02991                     self.ids.user_input.opacity = 0
02992                     self.record.energy_threshold = 1000
02993                     self.caprica_speak('User voice mode is now enabled. Click the button
    labeled record and then Speak into your microphone to begin your conversation.')
02994                 elif self.voice_enabled:
02995                     self.voice_enabled = False
02996                     self.voice_disabled = True
02997                     self.ids.user_input.opacity = 1
02998                     self.ids.view_port.opacity = 1
02999                     self.caprica_speak('User voice has been disabled. Type your response into
    the text box to begin your conversation.')
03000         except sr.UnknownValueError as a:
03001             if sys.platform.startswith('linux'):
03002                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
    speech_recognition.Recognizer.UnknownValueError: ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime
    ('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03003             elif sys.platform.startswith('win'):
03004                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
    speech_recognition.Recognizer.UnknownValueError: ' + str(a) + '\nDate - Time:' + str(datetime.datetime.now().strftime
    ('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error Logs.txt')
03005             return None
03006         except sr.RequestError as b:
03007             if sys.platform.startswith('linux'):
03008                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
    speech_recognition.Recognizer.RequestError: ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('
    %Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03009             elif sys.platform.startswith('win'):
03010                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
    speech_recognition.Recognizer.RequestError: ' + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('
    %Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error Logs.txt')
03011             return None
03012         except OSError as c:
03013             if sys.platform.startswith('linux'):
03014                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\nOSError:
    ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
    os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03015             elif sys.platform.startswith('win'):
03016                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\nOSError:
    ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
    C://SentienceFiles//Error Logs.txt')
03017             return None
03018         except IOError as d:
03019             if sys.platform.startswith('linux'):
03020                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\nIOError:
    ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
    os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03021             elif sys.platform.startswith('win'):
03022                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\nIOError:
    ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
    C://SentienceFiles//Error Logs.txt')
03023             return None
03024         except RuntimeError as e:
03025             if sys.platform.startswith('linux'):
03026                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
    RuntimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
    str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03027             elif sys.platform.startswith('win'):
03028                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
    RunetimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
    C://SentienceFiles//Error Logs.txt')
03029             return None
03030         except ValueError as f:
03031             if sys.platform.startswith('linux'):
03032                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
    ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
    str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03033             elif sys.platform.startswith('win'):
```

```
03034                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
      ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03035             return None
03036         except ImportError as g:
03037             if sys.platform.startswith('linux'):
03038                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
      ImportError: ' + str(g) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
      str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03039             elif sys.platform.startswith('win'):
03040                 self.__append_file('\n' + 'Function: set_enable_disable_voice ' + '\n
      ImportError: ' + str(g) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03041             return None
03042
03043
03044
03045     def print_files(self, path, filename):
03046         '''
03047         def print_files(self, path, filename)
03048             This function is called by the function
03049             self.open_print_dialog(self) function which is
03050             called by clicking on the 'Print' button on the
03051             menu bar. A new Popup() window is created
03052             which allows the user the ability to navigate to
03053             and select a specific file which they want to print.
03054             Once the user has selected that file they can click they
03055             'Print' button on the bottom bar of the Popup() window.
03056             Which then calls this function.
03057
03058
03059         Parameters:
03060         -----------
03061             param1 : self
03062                 Denotes it as being a member of SentienceScreen(Screen)
03063                 class.
03064
03065             param2 : path
03066                 This is the path to the file. Note:
03067                 including the file name is redundant. If the
03068                 selection tool for the 'Select file' Popup()
03069                 window function is re-written. It can return the
03070                 full path and not separate it.
03071
03072             param3 : filename
03073                 The name of the file to be printed. Note:
03074                 This is redundant see the param2 explanation.
03075
03076         Attributes
03077         ----------
03078             temp
03079                 The temp variable is a string variable.
03080                 This variable joins the path and filename
03081                 parameters to gain the absolute path of the
03082                 file to be printed.
03083                 temp = str(path) + str(filename) they're type
03084                 casted for safety.
03085
03086             path
03087                 The path variable stores the path to the
03088                  file that user has selected and wishes to print.
03089                  This is passed along with file name when the
03090                  user clicks the 'print' button on the bottom
03091                  bar of the PrintDialog() Popup() window.
03092
03093             filename
03094                 The filename variable stores the file name of the
03095                 file that user has selected and wishes to print.
03096                 This is passed along with the path when the
03097                 user clicks the 'print' button on the bottom
03098                 bar of the PrintDialog() Popup() window.
03099
03100             toBytes
03101                 toBytes is exactly what it sounds like. When Linux
03102                 users access this print_files(self, path, filename)
03103                 function. The path and file name are created as a single
03104                 string. Which is then converted to a bytes object for
03105                 printing. Note: This is redundant, you'll note in the
03106                 windows section of the code that I've simply used the
03107                 built in cast for the string class to encode the string
03108                 as it's passed to the native print function. That is to
03109                 say str.encode('') which returns the encoded string. I
03110                 could and should do that for the linux section as well.
03111
03112             import win32api
03113                 If the user is using a windows based operating
03114                 system. This is imported in that section. This
```

```
03115                 import gives us access to the native windows32
03116                 api print calls. Note: This is not being used
03117                 right now as I'm testing a new way of doing this
03118                 that is more pythonic than calling the windows Shell
03119                 directly. If this import statement exists outside of
03120                 this function the program will not run. Because it
03121                 will cause a fatal import error on Linux systems.
03122
03123             import win32print
03124                 If the user is using a windows based operating
03125                 system. This is imported in that section. This
03126                 import gives us access to the native windows32
03127                 api print calls. Note: This is not being used
03128                 right now as I'm testing a new way of doing this
03129                 that is more pythonic than calling the windows Shell
03130                 directly. If this import statement exists outside of
03131                 this function the program will not run. Because it
03132                 will cause a fatal import error on Linux systems.
03133
03134             lpr
03135                 This directly access the printer driver on a Linux based
03136                 operating system.
03137
03138             stdin
03139                 while creating the lpr object we set the stdin variable
03140                 to access the subprocess call to subprocess.PIPE. From
03141                 this call we're able to open and read in a file that's
03142                 contents will be piped to the variable for printing.
03143
03144         Members
03145         -------
03146             win32api.ShellExecute()
03147                 Executes a windows shell to directly call
03148                 the windows 32 api printer calls. Uses
03149                 win32print.GetDefaultPrinter() to return
03150                 and select the active printer.
03151
03152             win32print.GetDefaultPrinter()
03153                 This function is exactly what it sounds like.
03154                 It returns the default system printer and
03155                 when accessed and called as it is in this
03156                 function the default printer ID is returned
03157                 in the position of 'what printer do I send
03158                 this file to'.
03159
03160             lpr.stdin.write()
03161                 Takes the data stored in the variable toBytes pipes it
03162                 to the active printer.
03163
03164             subprocess.Popen()
03165                 Opens the active printer by directly accessing the
03166                 driver and then the default system printer.
03167
03168             subprocess.PIPE
03169                 Allows us to pipe the data in toBytes to the active
03170                 printer.
03171
03172             os.startfile('')
03173                 This is a pyhtonic command which opens the file and
03174                 then if told to via 'print' string, sends the
03175                 specific file to the default printer.
03176
03177         Private Members
03178         ---------------
03179             None
03180
03181         Returns
03182         -------
03183             return None
03184
03185         Exceptions
03186         ----------
03187             OSError
03188                 The OSError can occur due to numerous reasons.
03189                 What I'm primarily concerned with here however
03190                 is import statements, incompatible Operating
03191                 systems, and bad system calls. The exception
03192                 if it occurs is handled and logged in an error
03193                 log text file.
03194
03195             IOError
03196                 The IOError can occur due to many reasons.
03197                 My primary concern is file manipulation. The
03198                 improper opening/closing/writing to files. If
03199                 the exception occurs it's handled and logged; in
03200                 an error log text file.
03201
```

```
03202             RunTimeError
03203                 The RunTimeError error here is checking to make sure
03204                 that the chat bot doesn't die. Essentially I just need
03205                 to make sure that it completes and executes the python
03206                 text to speech functions in a manner that doesn't cause
03207                 a fatal exception. If something does occur the exception
03208                 will be handled and logged to an error log text file.
03209
03210             ValueError
03211                 Ensures that values passed to the chat bot are
03212                 appropriate. And if for some reason one isn't the
03213                 exception will be handled and logged to an error log
03214                 text file.
03215
03216             FileNotFoundError
03217                 This can occur in a variety of ways however my primary
03218                 concern is that file path the user selected is broken.
03219                 Resulting in an File Not Found error. If this occurs
03220                 it's handled and logged to an error file text log.
03221
03222             NameError
03223                 Again this can occur in a variety of ways but the
03224                 primary concern is that the conversion to bytes does
03225                 not take place or breaks some how due to wacky Unicode
03226                 characters. In which case the exception is handled and
03227                 logged to an error log text file.
03228         Notes
03229         -----
03230             When this function is called we check the users operating
03231             system. If sys.platform.startswith('linux') == True we know
03232             that the user is using a linux based operating system. In
03233             which case the appropriate if statements are executed.
03234
03235             Otherwise if sys.platform.startswith('win') == True then we
03236             know that the user is using a windows based operating system.
03237             In which case the appropriate if statements are executed.
03238
03239             For linux users the process is relatively straight forward.
03240             We directly access the printer driver in user/bin we
03241             determine the active printer. We then write the stream to
03242             said printer to actually print the file.
03243
03244             For windows users we have two methods though one is
03245             commented out. The active method is the pythonic version.
03246             os.startfile() We pass the file path and the 'print' string
03247             to let the function know that we mean to print the file at
03248             location path. It does the same thing the commented out
03249             section does it just cuts out those steps and uses pythons
03250             built in os library. Which preforms those steps behind the
03251             scenes.
03252         '''
03253         try:
03254             if sys.platform.startswith('linux'):
03255                 # temp = str(filename)
03256                 # temp = temp[2:-2]
03257                 # lpr = subprocess.Popen('/usr/bin/lpr', stdin = subprocess.PIPE)
03258                 # lpr.stdin.write(str.encode(temp))
03259                 os.startfile('lp "temp"')
03260                 self.ids.view_port.text = 'Printing will begin when program closes due to the GIL \n
      Blocking true multithreading.'
03261             elif sys.platform.startswith('win'):
03262                 # temp = str(path) + str(filename)
03263                 # import win32api
03264                 # import win32print
03265                 # win32api.ShellExecute(0, "printto", temp, '"%s"' % win32print.GetDefaultPrinter(), ".",
       0)
03266                 os.startfile(str.encode(temp), 'print')
03267         except OSError as a:
03268             if sys.platform.startswith('linux'):
03269                 self.__append_file('\n' + 'Function: print_files ' + '\nOSError: ' + str(a) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
       + '/.SentienceFiles/Error Logs.txt')
03270             elif sys.platform.startswith('win'):
03271                 self.__append_file('\n' + 'Function: print_files ' + '\nOSError: ' + str(a) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03272             return None
03273         except IOError as b:
03274             if sys.platform.startswith('linux'):
03275                 self.__append_file('\n' + 'Function: print_files ' + '\nIOError: ' + str(b) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
       + '/.SentienceFiles/Error Logs.txt')
03276             elif sys.platform.startswith('win'):
03277                 self.__append_file('\n' + 'Function: print_files ' + '\nIOError: ' + str(b) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03278             return None
```

```
03279         except RuntimeError as c:
03280             if sys.platform.startswith('linux'):
03281                 self.__append_file('\n' + 'Function: print_files ' + '\nRuntimeError: ' + str(
      c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03282             elif sys.platform.startswith('win'):
03283                 self.__append_file('\n' + 'Function: print_files ' + '\nRuntimeError: ' + str(
      c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03284             return None
03285         except ValueError as d:
03286             if sys.platform.startswith('linux'):
03287                 self.__append_file('\n' + 'Function: print_files ' + '\nValueError: ' + str(d)
       + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin
      ()) + '/.SentienceFiles/Error Logs.txt')
03288             elif sys.platform.startswith('win'):
03289                 self.__append_file('\n' + 'Function: print_files ' + '\nValueError: ' + str(d)
       + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03290             return None
03291         except FileNotFoundError as e:
03292             if sys.platform.startswith('linux'):
03293                 self.__append_file('\n' + 'Function: print_files ' + '\nFileNotFoundError: ' +
       str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03294             elif sys.platform.startswith('win'):
03295                 self.__append_file('\n' + 'Function: print_files ' + '\nFileNotFoundError: ' +
       str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03296             return None
03297         except NameError as f:
03298             if sys.platform.startswith('linux'):
03299                 self.__append_file('\n' + 'Function: print_files ' + '\nNameError: ' + str(f)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin(
      )) + '/.SentienceFiles/Error Logs.txt')
03300             elif sys.platform.startswith('win'):
03301                 self.__append_file('\n' + 'Function: print_files ' + '\nNameError: ' + str(f)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03302             return None
03303
03304
03305
03306     def create_dir(self, path):
03307         '''
03308         def create_dir(self, path)
03309             This function is called during the
03310             SentienceScreen().__init__() function.
03311             It creates a directory (folder) that
03312             will be used to store a series of files
03313             in.
03314
03315
03316         Parameters:
03317         -----------
03318             param1 : self
03319                 Denotes it as being a member of SentienceScreen(Screen)
03320                 class.
03321
03322             param2 : path
03323                 This is the path for the folder we're about
03324                 to create in the function self.create_dir(path)
03325
03326
03327         Attributes
03328         ----------
03329             path
03330                 The path variable stores the path to the
03331                 location where we will create a folder on
03332                 the users operating system. We will create
03333                 a series of required files when we call the
03334                 self.__create_files(path) function.
03335                 This path is based on the users operating system.
03336
03337
03338             Members
03339             -------
03340                 os.mkdir()
03341                     This function is called to access the systems native
03342                     directory creation process. On linux the command is
03343                     simply mkdir. Whereas on windows you're accessing
03344                     the win32 api and calling the C CREATE_DIRECTORY
03345                     binding function.
03346
03347         Private Members
03348         ---------------
03349             self.__create_files(path)
```

```
03350                 We call this function after we've created the
03351                 folder that we intend to store the required files in.
03352                 We pass one parameter to this function and it's path.
03353                 I've set the files names to be specific so All I need
03354                 to do is path + 'file name' inside the function
03355                 self.__create_files(path)
03356 
03357         Returns
03358         -------
03359             return None
03360 
03361         Exceptions
03362         ----------
03363             OSError
03364                 The OSError can occur due to numerous reasons.
03365                 What I'm primarily concerned with here however
03366                 is import statements, incompatible Operating
03367                 systems, and bad system calls. The exception
03368                 if it occurs is handled and logged in an error
03369                 log text file.
03370 
03371             IOError
03372                 The IOError can occur due to many reasons.
03373                 My primary concern is file manipulation. The
03374                 improper opening/closing/writing to files. If
03375                 the exception occurs it's handled and logged; in
03376                 an error log text file.
03377 
03378             RunTimeError
03379                 The RunTimeError error here is checking to make sure
03380                 that the chat bot doesn't die. Essentially I just need
03381                 to make sure that it completes and executes the python
03382                 text to speech functions in a manner that doesn't cause
03383                 a fatal exception. If something does occur the exception
03384                 will be handled and logged to an error log text file.
03385 
03386             ValueError
03387                 Ensures that values passed to the chat bot are
03388                 appropriate. And if for some reason one isn't the
03389                 exception will be handled and logged to an error log
03390                 text file.
03391 
03392             FileNotFoundError
03393                 This can occur in a variety of ways however my primary
03394                 concern is that file path the user selected is broken.
03395                 Resulting in an File Not Found error. If this occurs
03396                 it's handled and logged to an error file text log.
03397 
03398             NameError
03399                 Again this can occur in a variety of ways but the
03400                 primary concern is that the conversion to bytes does not
03401                 take place or breaks some how due to wacky Unicode
03402                 characters. In which case the exception is handled and
03403                 logged to an error log text file.
03404         Notes
03405         -----
03406             This function is called during the initialization of
03407             SentienceScreen() it's purpose is to create a folder.
03408 
03409             In this folder we store a number of text files which
03410             are created after the folder has been made; at which time
03411             another function is called from within self.create_dir(path)
03412             which then creates those text files.
03413         '''
03414         try:
03415             if sys.platform.startswith('linux'):
03416                 if os.path.isdir(path):
03417                     pass
03418                 elif not os.path.isdir(path):
03419                     os.mkdir(path)
03420                     self.__create_files(path)
03421             elif sys.platform.startswith('win'):
03422                 if os.path.isdir(path):
03423                     pass
03424                 elif not os.path.isdir(path):
03425                     os.mkdir(path)
03426                     self.__create_files(path)
03427         except IOError as a:
03428             if sys.platform.startswith('linux'):
03429                 self.__append_file('\n' + 'Function: create_dir ' + '\nIOError: ' + str(a) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
      + '/.SentienceFiles/Error Logs.txt')
03430             elif sys.platform.startswith('win'):
03431                 self.__append_file('\n' + 'Function: create_dir ' + '\nIOError: ' + str(a) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03432             return None
```

```
03433         except OSError as b:
03434             if sys.platform.startswith('linux'):
03435                 self.__append_file('\n' + 'Function: create_dir ' + '\nOSError: ' + str(b) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
      + '/.SentienceFiles/Error Logs.txt')
03436             elif sys.platform.startswith('win'):
03437                 self.__append_file('\n' + 'Function: create_dir ' + '\nOSError: ' + str(b) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03438             return None
03439         except FileNotFoundError as c:
03440             if sys.platform.startswith('linux'):
03441                 self.__append_file('\n' + 'Function: create_dir ' + '\nFileNotFoundError: ' +
      str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03442             elif sys.platform.startswith('win'):
03443                 self.__append_file('\n' + 'Function: create_dir ' + '\nFileNotFoundError: ' +
      str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03444             return None
03445         except FileExistsError as d:
03446             if sys.platform.startswith('linux'):
03447                 self.__append_file('\n' + 'Function: create_dir ' + '\nFileExistsError: ' +
      str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03448             elif sys.platform.startswith('win'):
03449                 self.__append_file('\n' + 'Function: create_dir ' + '\nFileExistsError: ' +
      str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03450             return None
03451
03452
03453
03454     def __create_files(self, path):
03455         '''
03456         def __create_files(self, path)
03457             This function is called during the
03458             SentienceScreen().__init__() function.
03459             From within the self.create_dir(self, path)
03460             function.
03461
03462         Parameters:
03463         -----------
03464             param1 : self
03465                 Denotes it as being a member of SentienceScreen(Screen)
03466                 class.
03467
03468             param2 : path
03469                 The path variable is passed to this function
03470                 from the self.create_dir(self, path) function
03471                 which is also the calling function for
03472                 self.__create_files(self, path).
03473
03474         Attributes
03475         ----------
03476             path
03477                 The path variable stores the path to the
03478                 location where we previously created a
03479                 folder. This is the same path that we will
03480                 use to create three text files.
03481                 Caprica_Statements.txt
03482                 User_Statements.txt
03483                 Error Logs.txt
03484                 We simply append those three file names to the
03485                 end of the passed path variable.
03486
03487         Members
03488         -------
03489             os.path.isfile('path to file')
03490                 We call this function to ensure that
03491                 the files we're attempting to create
03492                 don't already exist.
03493                 If os.path.isfile() == True then the
03494                 file(s) exist and we do nothing.
03495                 If os.path.isfile() == False then the
03496                 files do not exist and we create them.
03497
03498         Private Members
03499         ---------------
03500             None
03501
03502         Returns
03503         -------
03504             return None
03505
03506         Exceptions
03507         ----------
```

```
03508             OSError
03509                 The OSError can occur due to numerous reasons.
03510                 What I'm primarily concerned with here however
03511                 is import statements, incompatible Operating
03512                 systems, and bad system calls. The exception
03513                 if it occurs is handled and logged in an error
03514                 log text file.
03515
03516             IOError
03517                 The IOError can occur due to many reasons.
03518                 My primary concern is file manipulation. The
03519                 improper opening/closing/writing to files. If
03520                 the exception occurs it's handled and logged; in
03521                 an error log text file.
03522
03523             RunTimeError
03524                 The RunTimeError error here is checking to make sure
03525                 that the chat bot doesn't die. Essentially I just need
03526                 to make sure that it completes and executes the python
03527                 text to speech functions in a manner that doesn't cause
03528                 a fatal exception. If something does occur the exception
03529                 will be handled and logged to an error log text file.
03530
03531             ValueError
03532                 Ensures that values passed to the chat bot are
03533                 appropriate. And if for some reason one isn't the
03534                 exception will be handled and logged to an error
03535                 log text file.
03536
03537             FileNotFoundError
03538                 This can occur in a variety of ways however my primary
03539                 concern is that file path the user selected is broken.
03540                 Resulting in an File Not Found error. If this occurs
03541                 it's handled and logged to an error file text log.
03542
03543             NameError
03544                 Again this can occur in a variety of ways but the
03545                 primary concern is that the conversion to bytes does
03546                 not take place or breaks some how due to wacky Unicode
03547                 characters. In which case the exception is handled and
03548                 logged to an error log text file.
03549         Notes
03550         -----
03551             This function is called during the initialization of
03552             SentienceScreen() from within the
03553             self.create_dir(self, path) function. The purpose of
03554             self.__create_files(self, path) is to create  series of
03555             files which we will use to store.
03556
03557             1: Caprica_Statements : Stores all response from the
03558                                     chat bot.
03559
03560             2: User_Statements : Stores all responses from the user.
03561
03562             3: Error Logs : Stores any exceptions that occur with a
03563                             time date and calling function stamp.
03564
03565             4: Username + _Conversation : This file will be eventually
03566                                 created and stored to maintain a
03567                                 comprehensive list of all chat bot and user
03568                                 responses as they relate to each other.
03569         '''
03570         try:
03571             if sys.platform.startswith('linux'):
03572                 if os.path.isfile(path + 'User_Statements.txt'):
03573                     pass
03574                 elif not os.path.isfile(path + 'User_Statements.txt'):
03575                     with open(path + 'User_Statements.txt', 'w') as out:
03576                         pass
03577                 if os.path.isfile(path + 'Caprica_Statements.txt'):
03578                     pass
03579                 elif not os.path.isfile(path + 'Caprica_Statements.txt'):
03580                     with open(path + 'Caprica_Statements.txt', 'w') as out:
03581                         pass
03582                 if os.path.isfile(path + 'Error Logs.txt'):
03583                     pass
03584                 elif not os.path.isfile(path + 'Error Logs.txt'):
03585                     with open(path + 'Error Logs.txt', 'w') as out:
03586                         pass
03587             elif sys.platform.startswith('win'):
03588                 if os.path.isfile(path + 'User_Statements.txt'):
03589                     pass
03590                 elif not os.path.isfile(path + 'User_Statements.txt'):
03591                     with open(path + 'User_Statements.txt', 'w') as out:
03592                         pass
03593                 if os.path.isfile(path + 'Caprica_Statements.txt'):
03594                     pass
```

```
03595                 elif not os.path.isfile(path + 'Caprica_Statements.txt'):
03596                     with open(path + 'Caprica_Statements.txt', 'w') as out:
03597                         pass
03598                 if os.path.isfile(path + 'Error Logs.txt'):
03599                     pass
03600                 elif not os.path.isfile(path + 'Error Logs.txt'):
03601                     with open(path + 'Error Logs.txt', 'w') as out:
03602                         pass
03603         except IOError as a:
03604             if sys.platform.startswith('linux'):
03605                 self.__append_file('\n' + 'Function: __create_files ' + '\nIOError: ' + str(a)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin
      ()) + '/.SentienceFiles/Error Logs.txt')
03606             elif sys.platform.startswith('win'):
03607                 self.__append_file('\n' + 'Function: __create_files ' + '\nIOError: ' + str(a)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
      Logs.txt')
03608             return None
03609         except OSError as b:
03610             if sys.platform.startswith('linux'):
03611                 self.__append_file('\n' + 'Function: __create_files ' + '\nOSError: ' + str(b)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin
      ()) + '/.SentienceFiles/Error Logs.txt')
03612             elif sys.platform.startswith('win'):
03613                 self.__append_file('\n' + 'Function: __create_files ' + '\nOSError: ' + str(b)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
      Logs.txt')
03614             return None
03615         except FileNotFoundError as c:
03616             if sys.platform.startswith('linux'):
03617                 self.__append_file('\n' + 'Function: __create_files ' + '\nFileNotFoundError:
      ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03618             elif sys.platform.startswith('win'):
03619                 self.__append_file('\n' + 'Function: __create_files ' + '\nFileNotFoundError:
      ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03620             return None
03621         except FileExistsError as d:
03622             if sys.platform.startswith('linux'):
03623                 self.__append_file('\n' + 'Function: __create_files ' + '\nFileExistsError: '
      + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03624             elif sys.platform.startswith('win'):
03625                 self.__append_file('\n' + 'Function: __create_files ' + '\nFileExistsError: '
      + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03626             return None
03627
03628
03629
03630     def __append_file(self, words, path):
03631         '''
03632         def __append_file(self, words, path)
03633             This function is called every time the user
03634             and or the chat bot speaks. It Appends every
03635             every conversation to the appropriate file.
03636
03637
03638         Parameters:
03639         -----------
03640             param1 : self
03641                 Denotes it as being a member of SentienceScreen(Screen)
03642                 class.
03643
03644             param2 : words
03645                 Words is a string variable that contains
03646                 the response spoken by either the chat bot
03647                 or the user. This is the string that's appended
03648                 to the appropriate text file.
03649
03650             param3 : path
03651                 The path variable is passed to this function
03652                 from the self.create_dir(self, path) function
03653                 which is also the calling function for
03654                 self.__create_files(self, path).
03655
03656
03657         Attributes
03658         ----------
03659             path
03660                 The path variable here is a reference
03661                 to the absolute file path of a specific
03662                 file. This function is called every time a
03663                 response is entered by the user and generated
03664                 by the chat bot. The response are then appended
03665                 to User_Statements, Caprica_Statements respectively.
```

```
03666
03667         Members
03668         -------
03669             os.path.isfile('path to file')
03670                 We call this function to ensure that
03671                 the files we're attempting to manipulate
03672                 already exist. If os.path.isfile() == True
03673                 then the file exists and the data stored in
03674                 the words variable is appended to the end of
03675                 the file. If os.path.isfile() == False then
03676                 the file does not exist and we re-call the
03677                 function self.__create_files(self, path).
03678
03679         Private Members
03680         ---------------
03681            self.__create_files(self, path)
03682                This function is called only if one
03683                of the files required files has been deleted.
03684                This function will then write the file to
03685                the disk.
03686
03687
03688         Returns
03689         -------
03690             return None
03691
03692         Exceptions
03693         ----------
03694             OSError
03695                 The OSError can occur due to numerous reasons.
03696                 What I'm primarily concerned with here however
03697                 is import statements, incompatible Operating
03698                 systems, and bad system calls. The exception
03699                 if it occurs is handled and logged in an error
03700                 log text file.
03701
03702             IOError
03703                 The IOError can occur due to many reasons.
03704                 My primary concern is file manipulation. The
03705                 improper opening/closing/writing to files. If
03706                 the exception occurs it's handled and logged; in
03707                 an error log text file.
03708
03709             RunTimeError
03710                 The RunTimeError error here is checking to make sure
03711                 that the chat bot doesn't die. Essentially I just need
03712                 to make sure that it completes and executes the python
03713                 text to speech functions in a manner that doesn't cause
03714                 a fatal exception. If something does occur the exception
03715                 will be handled and logged to an error log text file.
03716
03717             ValueError
03718                 Ensures that values passed to the chat bot are
03719                 appropriate. And if for some reason one isn't the
03720                 exception will be handled and logged to an error log
03721                 text file.
03722
03723             FileNotFoundError
03724                 This can occur in a variety of ways however my primary
03725                 concern is that file path the user selected is broken.
03726                 Resulting in an File Not Found error. If this occurs
03727                 it's handled and logged to an error file text log.
03728
03729             NameError
03730                 Again this can occur in a variety of ways but the
03731                 primary concern is that the conversion to bytes does
03732                 not take place or breaks some how due to wacky Unicode
03733                 characters. In which case the exception is handled and
03734                 logged to an error log text file.
03735         Notes
03736         -----
03737             This function is called every time the user or
03738             the chat bot inputs/generates a response. That response
03739             is then appended to it's respective file.
03740
03741             1: User response: User_Statements.txt
03742
03743             2: Chat bot response: Caprica_Statements
03744         '''
03745         try:
03746             if sys.platform.startswith('linux'):
03747                 if os.path.isfile(path):
03748                     with open(path, 'a') as ap:
03749                         ap.write(words)
03750                 elif not os.path.isfile(path):
03751                     self.__create_files(path)
03752             elif sys.platform.startswith('win'):
```

```
03753                 if os.path.isfile(path):
03754                     with open(path, 'a') as ap:
03755                         ap.write(words)
03756                 elif not os.path.isfile(path):
03757                     self.__create_files(path)
03758         except IOError as a:
03759             if sys.platform.startswith('linux'):
03760                 self.__append_file('\n' + 'Function: __append_files ' + '\nIOError: ' + str(a)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin
      ()) + '/.SentienceFiles/Error Logs.txt')
03761             elif sys.platform.startswith('win'):
03762                 self.__append_file('\n' + 'Function: __append_files ' + '\nIOError: ' + str(a)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
      Logs.txt')
03763             return None
03764         except OSError as b:
03765             if sys.platform.startswith('linux'):
03766                 self.__append_file('\n' + 'Function: __append_files ' + '\nOSError: ' + str(b)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin
      ()) + '/.SentienceFiles/Error Logs.txt')
03767             elif sys.platform.startswith('win'):
03768                 self.__append_file('\n' + 'Function: __append_files ' + '\nOSError: ' + str(b)
      + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
      Logs.txt')
03769             return None
03770         except FileNotFoundError as c:
03771             if sys.platform.startswith('linux'):
03772                 self.__append_file('\n' + 'Function: __append_files ' + '\nFileNotFoundError:
      ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03773             elif sys.platform.startswith('win'):
03774                 self.__append_file('\n' + 'Function: __append_files ' + '\nFileNotFoundError:
      ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03775             return None
03776         except FileExistsError as d:
03777             if sys.platform.startswith('linux'):
03778                 self.__append_file('\n' + 'Function: __append_files ' + '\nFileExistsError: '
      + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03779             elif sys.platform.startswith('win'):
03780                 self.__append_file('\n' + 'Function: __append_files ' + '\nFileExistsError: '
      + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03781             return None
03782
03783
03784
03785     def write_logs(self):
03786         '''
03787         def write_logs(self)
03788             This function is called when the user clicks the
03789             'Write Logs' button that's located on the menu bar.
03790             It's represented by the pencil. The purpose of this
03791             function is to write the contents of self.master_log
03792             and self.__user_profile to a text file named after
03793             the current user.
03794
03795         Parameters:
03796         -----------
03797             param1 : self
03798                 Denotes it as being a member of SentienceScreen(Screen)
03799                 class.
03800
03801         Attributes
03802         ----------
03803             self.master_log
03804                 We write the contents of self.master_log to a text
03805                 file named after the current user. We also write
03806                 the contents of self.__user_profile to the text file.
03807
03808             self.username
03809                 The self.username variable stores the users
03810                 input username. We use this variable to name
03811                 the file generated by this function.
03812                 self.username + '_Conversation.txt'
03813
03814         Members
03815         -------
03816             os.path.isfile('path to file')
03817                 We call this function to ensure that
03818                 the files we're attempting to manipulate
03819                 don't already exist. If os.path.isfile() == True
03820                 then the file exists and will be over written.
03821                 If os.path.isfile() == False then
03822                 the file does not exist and we will write
03823                 the file normally.
```

```
03824
03825             self.create_dir(self, path)
03826                 We call this function to check to make sure
03827                 that the folder holding the required files
03828                 for this program already exists. If it does exist we
03829                 skip this if statement and write the file created by
03830                 this function. If it doesn't exist we call
03831                 self.create_dir(path) and re-create the folder so
03832                 that we can store the soon to be created file.
03833
03834         Private Members
03835         ---------------
03836             self.__create_files(self, path)
03837                 This function is called only if one
03838                 of the files required files has been deleted.
03839                 This function will then write the file to
03840                 the disk.
03841
03842             self.__user_profile
03843                 The dictionary variable self.__user_profile
03844                 contains a series of keys, Username, Sex, and
03845                 gender. This information is written to the start
03846                 of the file created by this function to clearly
03847                 state in text who the user is.
03848
03849         Returns
03850         -------
03851             return None
03852
03853         Exceptions
03854         ----------
03855             OSError
03856                 The OSError can occur due to numerous reasons.
03857                 What I'm primarily concerned with here however
03858                 is import statements, incompatible Operating
03859                 systems, and bad system calls. The exception
03860                 if it occurs is handled and logged in an error
03861                 log text file.
03862
03863             IOError
03864                 The IOError can occur due to many reasons.
03865                 My primary concern is file manipulation. The
03866                 improper opening/closing/writing to files. If
03867                 the exception occurs it's handled and logged; in
03868                 an error log text file.
03869
03870             RunTimeError
03871                 The RunTimeError error here is checking to make sure
03872                 that the chat bot doesn't die. Essentially I just need
03873                 to make sure that it completes and executes the python
03874                 text to speech functions in a manner that doesn't cause
03875                 a fatal exception. If something does occur the exception
03876                 will be handled and logged to an error log text file.
03877
03878             ValueError
03879                 Ensures that values passed to the chat bot are
03880                 appropriate. And if for some reason one isn't the
03881                 exception will be handled and logged to an error
03882                 log text file.
03883
03884             FileNotFoundError
03885                 This can occur in a variety of ways however my
03886                 primary concern is that file path the user selected
03887                 is broken. Resulting in an File Not Found error. If this
03888                 occurs it's handled and logged to an error file text log.
03889
03890             NameError
03891                 Again this can occur in a variety of ways but the
03892                 primary concern is that the conversion to bytes does not
03893                 take place or breaks some how due to wacky Unicode
03894                 characters. In which case the exception is handled
03895                 and logged to an error log text file.
03896         Notes
03897         -----
03898             This function is called when the user clicks the
03899             'Write Logs' button that's located on the menu bar.
03900             It's represented by the pencil. The purpose of this
03901             function is to write the contents of self.master_log
03902             and self.__user_profile to a text file named after
03903             the current user.
03904
03905             The first thing that we do when this function is
03906             check the users operating system. If
03907             sys.platform.startswith('linux') == True
03908             then the user is running a linux based operating
03909             system and the appropriate if statements are
03910             executed.
```

```
03911
03912             Other wise if sys.platform.startswith('win') == False
03913             then the user is running a windows based operating system
03914             and the appropriate if statements are executed.
03915
03916             We then ensure that the directory created when the program
03917             first started exists. If it does not we re-create it. We
03918             then have to re-create the files that were stored in that
03919             folder.
03920
03921             After that we create a new file naemd after the current
03922             user self.username + '_Conversation.txt'. We then write
03923             the contents of self.__user_profile and self.master_log
03924             to that file.
03925         '''
03926         try:
03927             if sys.platform.startswith('linux'):
03928                 if not os.path.isdir('/home/' + str(os.getlogin()) + '/.SentienceFiles/'):
03929                     self.create_dir('/home/' + str(os.getlogin()) + '/.SentienceFiles/')
03930                     if not os.path.isfile('/home/' + str(os.getlogin()) + '
      /.SentienceFiles/Caprica_Statements.txt'):
03931                         self.__create_files('/home/' + str(os.getlogin()) + '
      /.SentienceFiles/')
03932                 else:
03933                     temp = '/home/' + str(os.getlogin()) + '/.SentienceFiles/' + self.
      username + '_Conversation.txt'
03934                     with open(temp, 'w') as out:
03935                         out.write('Username: ' + str(self.user_profile[1]) + '\nAge: ' + str(
      self.user_profile[2]) + '\nSex: ' + str(self.user_profile[3]) + '\n' + self.
      master_log)
03936             elif sys.platform.startswith('win'):
03937                 if not os.path.isdir('C://SentienceFiles//'):
03938                     self.create_dir('C://SentienceFiles//')
03939                     if not os.path.isfile('C://SentienceFiles//Caprica_Statements.txt'):
03940                         self.__create_files('C://SentienceFiles//')
03941                 else:
03942                     temp = 'C://SentienceFiles//' + self.username + '_Conversation.txt'
03943                     with open(temp, 'w') as out:
03944                         out.write('Username: ' + str(self.user_profile[1]) + '\nAge: ' + str(
      self.user_profile[2]) + '\nSex: ' + str(self.user_profile[3]) + '\n' + self.
      master_log)
03945         except IOError as a:
03946             if sys.platform.startswith('linux'):
03947                 self.__append_file('\n' + 'Function: write_files ' + '\nIOError: ' + str(a) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
       + '/.SentienceFiles/Error Logs.txt')
03948             elif sys.platform.startswith('win'):
03949                 self.__append_file('\n' + 'Function: write_files ' + '\nIOError: ' + str(a) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03950             return None
03951         except OSError as b:
03952             if sys.platform.startswith('linux'):
03953                 self.__append_file('\n' + 'Function: write_files ' + '\nOSError: ' + str(b) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
       + '/.SentienceFiles/Error Logs.txt')
03954             elif sys.platform.startswith('win'):
03955                 self.__append_file('\n' + 'Function: write_files ' + '\nOSError: ' + str(b) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
03956             return None
03957         except FileNotFoundError as c:
03958             if sys.platform.startswith('linux'):
03959                 self.__append_file('\n' + 'Function: write_files ' + '\nFileNotFoundError: ' +
       str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03960             elif sys.platform.startswith('win'):
03961                 self.__append_file('\n' + 'Function: write_files ' + '\nFileNotFoundError: ' +
       str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03962             return None
03963         except FileExistsError as d:
03964             if sys.platform.startswith('linux'):
03965                 self.__append_file('\n' + 'Function: write_files ' + '\nFileExistsError: ' +
      str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
03966             elif sys.platform.startswith('win'):
03967                 self.__append_file('\n' + 'Function: write_files ' + '\nFileExistsError: ' +
      str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
03968             return None
03969
03970
03971
03972     def open_print_file_dialog(self):
03973         '''
03974         def open_print_file_dialog(self): calls self.open_print_file_dialog
```

```
03975         when the user clicks on the "Print" button on the menu bar.
03976
03977         content = PrintDialog(print_files = self.print_files,
03978         Cancel = self.dismiss_popup) sets the ObjectProperty(s)
03979         of PrintDialog class to reference the local and or instance
03980         variables, functions (in this case)
03981
03982         self._popup = Popup(title = "Print File", content = content,
03983         size_hint = (0.9, 0.9)) create Popup, set title to 'Print File',
03984         content = content (PrintDialog), size_hint 90% width,height
03985
03986         self._popup.open() : calls function to open the popup
03987         '''
03988         content = PrintDialog(print_files = self.print_files, Cancel = self.
      dismiss_popup)
03989         self._popup = Popup(title = "Print File", content = content, size_hint = (0.9, 0.9))
03990         self._popup.open()
03991
03992
03993
03994     def dismiss_popup(self):
03995         '''
03996         def dismiss_popup(self): is called when the user clicks
03997         the Cancel button on the Popup
03998
03999         self._popup.dismiss() : calls the built in to dismiss the Popup
04000
04001         '''
04002         self._popup.dismiss()
04003
04004
04005
04006     def on_mouse_pos(self, instance, pos):
04007         '''
04008         def on_mouse_pos(self, instance, pos):
04009             This function is called everytime that the user moves
04010             the mouse. It checks to see if the mouse is colliding
04011             (hitting) any of the widgets on the menu bar. In this
04012             case, I focus on the buttons. If the mouse touches
04013             any of the buttons a tooltip is created and displayed
04014             where the mouse was located; explaining what that
04015             particular button does.
04016
04017
04018         Parameters:
04019         -----------
04020             param 1: self
04021                 Denotes this as being a member of SentienceScreen()
04022
04023             param 2: instance
04024                 Returns the current "instance" of the mouse. Similar
04025                 to coordinates in that it refers to "This current
04026                 position". If the mouse moves again its instance has
04027                 changed.
04028
04029             param 3: pos
04030                 The current coordinates of the mouse as it relates
04031                 to the window.
04032
04033
04034         Attributes
04035         ----------
04036             colliding_computer = self.ids.select_os.collide_point(*pos)
04037                 colliding_computer stores the collision point (the
04038                 coordinates of the "Select OS" button).
04039
04040             colliding_record = self.ids.record_user.collide_point(*pos)
04041                  colliding_record stores the collision point (the
04042                  coordinates of the "Record user" button).
04043
04044             colliding_voice = self.ids.voice_enable_disable.collide_point(*pos)
04045                  colliding_voice stores the collision point (the
04046                  coordinates of the "Enable/disable voice" button).
04047
04048             colliding_audio = self.ids.audio_enable_disable.collide_point(*pos)
04049                  colliding_audio stores the collision point (the
04050                  coordinates of the "Enable/Disable audio" button).
04051
04052             colliding_eraser = self.ids.erase_text_button.collide_point(*pos)
04053                  colliding_eraser stores the collision point (the
04054                  coordinates of the "Erase text" button).
04055
04056             colliding_pencil = self.ids.write_file_button.collide_point(*pos)
04057                  colliding_computer stores the collision point (the
04058                  coordinates of the "Write Logs" button).
04059
04060             colliding_printer = self.ids.print_logs.collide_point(*pos)
```

```
04061                colliding_computer stores the collision point (the
04062                coordinates of the "Print logs" button).
04063
04064            self.ids.select_os
04065                This is a reference to the select_os Button widget.
04066
04067            self.ids.record_user
04068                This is a reference to the record_user Button widget.
04069
04070            self.ids.voice_enable_disable
04071                This is a reference to the voice_enable_disable
04072                Button widget.
04073
04074            self.ids.audio_enable_disable
04075                This is a reference to the audio_enable_disable
04076                Button widget.
04077
04078            self.ids.erase_text_button
04079                This is a reference to the erase_text_button
04080                Button widget.
04081
04082            self.ids.write_file_button
04083                This is a reference to the write_file_button
04084                Button widget.
04085
04086            self.ids.print_logs
04087                This is a reference to the print_logs Button widget.
04088
04089            self.tooltip_open
04090                This is a member of SentienceScreen(). This is how
04091                we determine if a tooltip is currently open. If this
04092                is open we then know we need to close it and set it
04093                to self.tooltip_open = False
04094
04095            self.tooltip.pos
04096                This is a member of the ToolTipLabel widget. We simply
04097                set (or reset by setting it) the current position of this
04098                widget to the position of the instance of the pointer
04099                which collided with this calling function. I.e,.
04100                If the mouse collides wit the select_os button
04101                then we use that exact collision point to set
04102                the position of this widget and then add the tooltip
04103                at that position.
04104
04105
04106        Members
04107        -------
04108            self.ids.select_os.collide_point(*pos)
04109                Is called when the current instance and position of the
04110                mouse collide (touch/hit) the select_os button widget.
04111
04112            self.ids.record_user.collide_point(*pos)
04113                 Is called when the current instance and position of the
04114                 mouse collide (touch/hit) the record_user button widget.
04115
04116            self.ids.voice_enable_disable.collide_point(*pos)
04117                 Is called when the current instance and position of the
04118                 mouse collide (touch/hit) the voice_enable_disable
04119                 button widget.
04120
04121            self.ids.audio_enable_disable.collide_point(*pos)
04122                 Is called when the current instance and position of the
04123                 mouse collide (touch/hit) the audio_enable_disable
04124                 button widget.
04125
04126            self.ids.erase_text_button.collide_point(*pos)
04127                 Is called when the current instance and position of the
04128                 mouse collide (touch/hit) the erase_text_button button
04129                 widget.
04130
04131            self.ids.write_file_button.collide_point(*pos)
04132                 Is called when the current instance and position of the
04133                 mouse collide (touch/hit) the write_file_button button
04134                 widget.
04135
04136            self.ids.print_logs.collide_point(*pos)
04137                 Is called when the current instance and position of the
04138                 mouse collide (touch/hit) the print_logs button widget.
04139
04140            self.get_root_window()
04141                This function applies to the root window. It's called
04142                as a check when the users access the tooltips. The
04143                check preformed ensures that if the users moves the
04144                mouse out of the programs window the tooltip widget
04145                is destroyed.
04146
04147            self.set_tooltip_text(text)
```

```
04148                 We call this function to set the tooltip text.
04149                 We do this each time a tooltip is created but we only
04150                 change the text based on the widget that the mouse
04151                 collided with. We don't want the user to see "Select Os"
04152                 when they collide with the print_logs button
04153                 when they should see "Print file".
04154
04155             self.display_tooltip(*args)
04156                 We finally call this function actually
04157                 add a new label widget, which is our tooltip,
04158                 to the screen.
04159
04160             # Todo: update documentation to reflect current status.
04161         Private Members
04162         ---------------
04163             None
04164
04165         Returns
04166         -------
04167             return None
04168
04169         Exceptions
04170         ----------
04171            None
04172
04173         Notes
04174         -----
04175             this function is called whenever the user moves his or her
04176             mouse. It only ever "does something" when the mouse collides
04177             with a widget listed in the conditional statements. In this
04178             case, when the users mouse touches (collides) with one of
04179             the buttons on the menu bar. When that happens the if
04180             statements are checked and we determine which widget
04181             the mouse has collided with.
04182
04183             Once we've determined what widget the users mouse has
04184             collided with. We then set self.tooltip_open = True
04185             We set the position of the ToolTipLabel to the
04186             position of the users mouse when that mouse collided
04187             with that specific widget.
04188
04189             We then specify what text we want the tooltip to
04190             display as it relates to that specific widget.
04191
04192             Finally we call the function to create and add that
04193             widget to the screen.
04194         '''
04195         if not self.get_root_window():
04196             return
04197         colliding_record = self.ids.record_user.collide_point(*pos)
04198         colliding_settings = self.ids.open_settings.collide_point(*pos)
04199         if colliding_record and self.tooltip_open == False:
04200             self.current_conversation = ''
04201             if len(self.ids.view_port.text) > 0:
04202                 self.current_conversation = self.ids.view_port.text
04203             self.tooltip_open = True
04204             self.tooltip.pos = pos
04205             self.set_tooltip_text('Record')
04206             self.ids.view_port.text = ("This is the record button. "
04207                                        "Click this button to speak with your"
04208                                        " microphone, after you enable the voice"
04209                                        " option.")
04210             self.display_tooltip()
04211
04212
04213
04214     def display_tooltip(self, *args):
04215         '''
04216         display_tooltip(self, *args):
04217
04218         Parameters
04219         ----------
04220             param1 : self
04221                 Denotes this function as being a member of
04222                 self.SentienceScreen().
04223
04224             param2 : *args
04225                 Can take a list, array dict, etc.. of
04226                 arguments. This relates to the specific position and
04227                 instance of the pointer (mouse) when this function is
04228                 called.
04229         Attributes
04230         ----------
04231             self.tooltip
04232                 self.tooltip is the widget ToolTipLabel declared in
04233                 the kv design language. This is the tooltip widget
04234                 that we use to display the text which describes the
```

```
04235                 buttons the user hovers over.
04236             Members
04237             -------
04238                 Window
04239                     The window member relates to the kivy Window.
04240                     The Window is the main active root widget.
04241                     This should not be confused with root_widget.
04242                     The root widget that Window refers to is the
04243                     windowing system its self which is default and
04244                     separate from any user generated widgets.
04245
04246                 Window.add_widget()
04247                     When this is called we add a new widget
04248                     to the main active window. In this case
04249                     we're adding a Label widget which contains
04250                     descriptive text about the specific button
04251                     that the user is hovering over when this function
04252                     is called.
04253
04254                 Clock
04255                     This is the kivy clock, not the system clock.
04256                     This handles all of the frames, callbacks and events
04257                     in a kivy program. That is to say that this is what makes
04258                     everything work in that it calls things rhythmically and
04259                     prevents any thing from occurring concurrently witch could
04260                     break the program. It also has other uses such as registering
04261                     function calls that will occur at or during specific intervals.
04262
04263                 Clock.schedule_once(event, time)
04264                    Clock.schedule_once() is a way for us to
04265                    call a specific function once (not recursively,
04266                    or repetitively). This function call requires
04267                    an event, such as the calling of a function, and
04268                    a time frame, this time frame dictates when the
04269                    event occurs. In our case we call the event
04270                    five seconds after it's been registered here.
04271                    Or to be more accurate we call it five frames
04272                    after. Due to the way the kivy clock functions
04273                    the amount of time that this is executed in will
04274                    not always occur at the same time for a variety of
04275                    reasons. In actuality on most systems the call
04276                    will occur around .5 seconds after the event has been
04277                    registered. This function is used to call the
04278                    SentienceScreen().close_tooltip() function which
04279                    removes the tooltip from the screen.
04280
04281                 close_tooltip()
04282                     self.close_tooltip is a member of SentienceScreen().
04283                     We call this function to remove the tooltip from the
04284                     screen. It's called with the clock event.
04285             Private Members
04286             ---------------
04287                 None
04288             Returns
04289             -------
04290                 None
04291             Exceptions
04292             ----------
04293                 None
04294             Notes
04295             -----
04296                 I've outlined what this function does
04297                 fairly well in the above comments. But,
04298                 an overview is this. This function is called
04299                 when the user hovers their mouse over a button
04300                 on the menu bar (ActionBar). That specific
04301                 instance of the pointer (mouse) is then passed
04302                 to *args. We then add the ToolTipLabel Widget
04303                 to that button after the specified amount of
04304                 time.
04305             '''
04306             Window.add_widget(self.tooltip)
04307             Clock.schedule_once(self.close_tooltip, 11)
04308
04309
04310
04311         def close_tooltip(self, dt):
04312             '''
04313             close_tooltip(self, dt)
04314
04315             Parameters
04316             ----------
04317                 param1 : self
04318                     self denotes that this is a member of SentienceScreen().
04319
04320                 param2 : dt
04321                     The dt parameter is a float (double) value. It refers
```

```
04322                to a time. So in our case we supply the number 5 to
04323                this parameter when this function is called in
04324                self.display_tooltip(). The number 5 refers to
04325                milliseconds/seconds/frames. The time at which
04326                this function is called will be different from
04327                system to system but will not exceed 5 seconds.
04328
04329        Attributes
04330        ----------
04331            self.tooltip
04332                tooltip is a reference to the ToolTipLabel Widget
04333                in the kv design language. This the instantiated and
04334                mutable object of that widget.
04335
04336            self.tooltip_open
04337                We use tooltip_open to check whether or not the tooltip
04338                widget is currently "open", in other words, in use. If
04339                tooltip_open == True then we know that the tooltip
04340                widget is currently in use and we can close it. If
04341                it's False we know it's not in use and that it can
04342                be opened to display information about a widget on
04343                the menu bar (ActionBar).
04344            Window
04345                The window member relates to the kivy Window.
04346                The Window is the main active root widget.
04347                This should not be confused with root_widget.
04348                The root widget that Window refers to is the
04349                windowing system its self which is default and
04350                separate from any user generated widgets.
04351        Members
04352        --------
04353            Window.remove_widget(self.tooltip)
04354                This allows us to remove (delete) a widget
04355                from the current active window (widget). Remember
04356                this refers to the windowing system and the main
04357                window. That is to say that we can use this to
04358                remove a user created widget from the MainWindow.
04359                In this case we use it to remove SentienceScreen.tooltip.
04360                self.tooltip is the only parameter supplied to this function
04361                call as it's the only widget that we remove.
04362
04363        Private Members
04364        ---------------
04365            None
04366        Returns
04367        -------
04368            None
04369        Exceptions
04370        ----------
04371            None
04372        Notes
04373        -----
04374            I've outlined what this function does fairly well in
04375            the above comments. But, an overview of the function
04376            is this.
04377
04378            We call self.close_tooltip(event, dt) with the kivy
04379            Clock.schedule_once(event, dt) function. We only
04380            call self.close_tooltip if self.tooltip_open == True.
04381
04382            Calling this function allows us to remove the tooltip
04383            (Label) with descriptive text from the screen.
04384        '''
04385        self.tooltip_open = False
04386        if len(self.current_conversation) > 0:
04387            self.ids.view_port.text = self.current_conversation
04388        elif len(self.current_conversation) <=0:
04389            self.ids.view_port.text = ''
04390        Window.remove_widget(self.tooltip)
04391
04392
04393
04394    def set_tooltip_text(self, text):
04395        '''
04396        set_tooltip_text(self, text)
04397
04398        Parameters
04399        ----------
04400            param1 : self
04401                self denotes this function as being a member of
04402                SentienceScreen().
04403
04404            param2 : text
04405                text is a string variable which holds a string
04406                passed to it by the developer. In this case,
04407                the string contains descriptive text about
04408                each specific button on the menu bar (Action
```

```
04409                Bar). It's called from within the self.on_mouse_pos()
04410                function; and relates to each specific position. In
04411                other words this function is called and each time
04412                the "text" parameter contains different text for
04413                each different button.
04414            Attributes
04415            ----------
04416                self.tooltip
04417                    Refers to the ToolTipLabel in the kv design
04418                    language. This is the mutable instantiated
04419                    object of that widget. We use this to add/remove
04420                    the widget to and from the screen. As well as
04421                    changing its text. We can also do whatever else
04422                    to the widget that's possible with this object.
04423
04424            Members
04425            -------
04426                self.tooltip.text
04427                    The way that we change the text of the label,
04428                    tooltip is both a function and a property.
04429                    It's a property and it's set but it's set by
04430                    a function call. We set the text of tooltip
04431                    by saying self.tooltip.text = 'insert text'.
04432                    We use this property to set and change the
04433                    text for each button on the menu bar (Action
04434                    Bar).
04435            Private Members
04436            ---------------
04437                None
04438            Exceptions
04439            ----------
04440                None
04441            Returns
04442            -------
04443                None
04444            Notes
04445            -----
04446                This function is really straight forward.
04447                Every time the user hovers his or her pointer
04448                (mouse) over a button on the menu bar (Action
04449                Bar) a tooltip is created and added to the screen.
04450                Before the tooltip is added to the screen we change
04451                its text so that it contains information specific to
04452                the button that the mouse just touched.
04453            '''
04454            self.tooltip.text = text
04455
04456
04457
04458        def caprica_timer(self, _time):
04459            '''
04460            def caprica_timer(self, _time)
04461
04462            Parameters
04463            ----------
04464                param1 : self
04465                    Denotes this as being a member of the SentienceScreen()
04466                    class.
04467                param2 : _time
04468                    Can be either double or of type int. I'm using it
04469                    as an integer by supplying it with a whole part. This
04470                    variable dictates how long the timer which is this
04471                    function runs.
04472            Attributes
04473            ----------
04474                mins
04475                    This variable stores the number of minutes that
04476                    this timer function will run. mins is displayed and
04477                    along with secs ticks down to reflect the amount of
04478                    time that this function will run. Though the user can't
04479                    see the visual display.
04480                secs
04481                    This variable the number of seconds that this timer
04482                    function will run. secs is displayed and along with
04483                    mins ticks down to reflect the amount of time that
04484                    this function will run. Though the user can't see
04485                    the visual display.
04486                timeformat
04487                    timeformat is the format of how the time will apear
04488                    to the user when it's printed to the console. It
04489                    looks like this. If you supply, 168 to this function
04490                    it would output 2:48. Though the user can't see this
04491                    visual timer.
04492            Members
04493            -------
04494                time.sleep(integer)
04495                    We call time.sleep() to ensure that the timer
```

```
04496                 only counts down 1 second at a time and that it
04497                 doesn't interfere with any other active thread.
04498             divmod()
04499                 This is a builtin python function which returns
04500                 the quotient and remainder of the two numbrs
04501                 whic are supplied to it; in this case mins, secs.
04502             str().format()
04503                 This is a member of the built in python string class.
04504                 It formats teh string to look however you set it. In
04505                 our case we format the ticker display to print out
04506                 2:48 if supplied with 168, if it were 120 it would
04507                 look like 2:00.
04508             self.check_timer(_time)
04509                 This is a member of the SentienceScreen() class.
04510                 We call this function to ensure that _time is
04511                 not less than or equal to zero if it is we terminate
04512                 both self.check_timer and self.caprica_timer().
04513             self.notification_widget.foreground_color
04514                 This is a member of the SentienceScreen() class. It's
04515                 one of our TextInput widgets. We use this to change the
04516                 foreground color, which is the color of the text. To
04517                 reflect the active status of the program. If the chatbot
04518                 is about to generate a response for the user the color
04519                 of the text is changed to red. If the chatbot has just
04520                 finished generating a response to the user the color
04521                 of the text is blue.
04522             self.notification_widget.text
04523                 This is a member of the SentienceScreen() class. We
04524                 use this to set the text property of the
04525                 self.notification_widget which is one of our
04526                 TextInput widgets. We do this to reflect the current
04527                 status of the program. If the chatbot is about to
04528                 generate a response for the user we change the text
04529                 to '...Thinking...' and set the color of the text to
04530                 red. If the chatbot has just finished generating a
04531                 response to the user we set the text to '...Inactive...'
04532                 and change the color of the text to blue.
04533             kivy.utils.get_color_from_hex()
04534                 This is a member of the kivy.utils() class. We call
04535                 this function to convert a hexadecimal string to
04536                 an integer or double value that (automaticaly double)
04537                 that can be interpereted by the TextInput widget
04538                 as an appropriate and existing color code. Kivy uses
04539                 the opengl method setting colors and it's easier
04540                 for me to work with hex then it is for me to determine
04541                 the rgba-opnegl equivelant.
04542         Private Members
04543         ---------------
04544             None
04545         Exceptions
04546         ----------
04547             None
04548         Returns
04549         -------
04550             None
04551         Notes
04552         -----
04553             This function is not currently being used. But,
04554             an explanation of it's use is as follows. The developer
04555             supplies a number to the _time variable. This number
04556             represents the time that this function will run.
04557
04558             This function should run as an independent thread which
04559             constantly ticks down until _time = 0. While it ticks
04560             down it should also flash the text '...Thinking...'
04561             until the function terminates when it sets the text
04562             to '...Inactive...'
04563         '''
04564         while _time:
04565             mins, secs = divmod(_time, 60)
04566             timeformat = '{:02d}:{:02d}'.format(mins, secs)
04567             time.sleep(1)
04568             _time -= 1
04569             self.ids.notification_widget.foreground_color = kivy.utils.get_color_from_hex('FF0000')
04570             self.ids.notification_widget.text = '..Thinking..'
04571         if self.check_timer(_time):
04572             self.ids.notification_widget.text = '...Inactive...'
04573             self.ids.notification_widget.foreground_color = kivy.utils.get_color_from_hex('00FFFF')
04574
04575
04576
04577     def start_timer_thread(self, _time):
04578         '''
04579         start_timer_thread(self, _time)
04580
04581         Parameters
04582         ----------
```

```
04583             self
04584                 self denotes this function as being a member of
04585                 SentienceScreen().
04586             _time
04587                 _time is a double variable which contains a number.
04588                 That numbers refers to a specific time value.
04589                 For instance, if we pass 20 to _time it means
04590                 five seconds. We use _time to run an event for _time
04591                 length.
04592         Attributes
04593         ----------
04594         self.ids.notification_widget
04595             Refers to the notification_widget TextInput Widget in
04596             the kv design language.
04597
04598         target
04599            target is an attribute of the threading.Thread class.
04600            We use that to register our event, which in this case
04601            is the function self.caprica_timer.
04602         args
04603             args is an attribute of the threading.Thread class.
04604             Its a tupple of arguments which will store the parameters
04605             of the event that target =. In this case args = _time
04606             which again holds a numerical value which refers to the time
04607             that the function self.caprica_timer will run. To be more
04608             accurate it's the time that self.caprica_timer will count
04609             down from.
04610         Members
04611         -------
04612             self.notification_widget.opacity
04613                 This both a function and a property. We use
04614                 this to set the opacity of the notification_widget
04615                 TextInput widget which is in the kv design language.
04616                 When opacity = 1 it's visible to the user. When
04617                 opactiy = 0 it's invisible to the user.
04618
04619             threading.Thread()
04620                 Thread is a member of the threading class. We use this
04621                 to decalre, initialize and run a new thread.
04622             threading.Thread.start()
04623                 start() is a member of the threading.Thread class. This
04624                 is what we use to actually start or run our newly
04625                 created thread. Which in this case is
04626                 self.caprica_timer().
04627         Private Members
04628         ---------------
04629             None
04630         Exceptions
04631         ----------
04632             None
04633         Returns
04634         -------
04635             None
04636         Notes
04637         -----
04638             We call this function to start a new thread to run
04639             the function self.caprica_timer. It's run as a seperate
04640             thread to prevent the user from thinking that the program
04641             is crashing. It's also much more efficent to do it this way.
04642
04643             Unfortunately, on windows operating systems threading and
04644             multiprocessing has the effect of launching a new python
04645             interpreter in the form of a new window which quickly pops
04646             up and vanishes from the screen which could cuase fear in the
04647             user.
04648
04649             However, this is not a bug, it's an intended feature. Python
04650             is neither meant for nor truly not meant for multithreading.
04651             However, the GIL or Global Interpereter Lock prevents true
04652             multithreading from occuring to prevent huge memory leaks
04653             and unsafe practices. There is unfortunately no way around
04654             this windowing effect. But, it's okay because aside from it
04655             being a minor annoyance it's not an actual issue.
04656
04657             Essentially, this function is called and it sets the opacity
04658             of notification_widget TextInput Widget to 1; rendering it
04659             visible to the user. A new thread is then created and executed
04660             which enables the notification_widget to display "..Thinking.."
04661             while the bot searches its database for an answer.
04662
04663             Users may or may not see this notification based on the
04664             "Magic Window" that pops up and based on the amount of time
04665             that it takes the bot to locate an appropriate response.
04666         '''
04667         threading.Thread(target = self.caprica_timer, args = (_time,)).start()
04668
04669
```

```
04670
04671     def check_timer(self, _time):
04672         '''
04673         check_timer(self, _time)
04674
04675         Parameters
04676         ----------
04677             param1 : self
04678                 self denotes this function as being a member of
04679                 SentienceScreen().
04680
04681             param2 : _time
04682                 _time is a double variable which contains a number.
04683                 This number is used in self.caprica_timer as a countdown.
04684                 This function monitors that countdown and ensures that
04685                 when time is <= 0 the while loop in self.caprica_timer
04686                 is broken. We also use this to know when to
04687                 disable/enable some other features in that function.
04688                 More information about self.caprica_timer can be found
04689                 in it's comments. For our purpose here we check _time
04690                 to see if it's <= 0 if it is we return True if
04691                 _time is > 0 we return False.
04692         Attributes
04693         ----------
04694             _time
04695                 See above information in Paraemters section.
04696         Members
04697         -------
04698             None
04699         Private Members
04700         ---------------
04701             None
04702         Exceptions
04703         ----------
04704             None
04705         Returns
04706         -------
04707             True
04708                 We return True if _time is <= 0.
04709             False
04710                 We return False if _time is > 0.
04711         Notes:
04712             This function is called during self.caprica_timer
04713             to check the variable _time. If the number stored in
04714             the variable _time is less than or equal to 0 we
04715             return True. If the number stored in _time is
04716             greater than 0 we return False.
04717         '''
04718         if _time <= 0:
04719             return True
04720         elif _time > 0:
04721             return False
04722
04723
04724
04725     def get_caprica_response(self):
04726         '''
04727         get_caprica_response(self)
04728
04729         Attributes
04730         ----------
04731             my_timer
04732                 my_timer is the decleration and intialization of
04733                 threading.Thread(target = event, args = (params)).start()
04734                 This one line code creates and starts a new thread. This
04735                 thread allows us to use the self.caprica_timer() function.
04736             target
04737                 This variable is a member of the threading.Thread()
04738                 class. It's used to register the passed event. Then
04739                 call said event which is in this case the function
04740                 self.start_timer_thread; which in turn calls the
04741                 function self.caprica_timer.
04742             args
04743                 This variable is a member of the threading.Thread()
04744                 class. It's used to store the parameters of the
04745                 event thats passed to the target member of the
04746                 threading.Thread() class. In this case we pass
04747                 a number to it. This number is a double variable
04748                 and refers to the amount of time that will be used
04749                 in the self.caprica_timer function.
04750             response
04751                 The response variable is used to store the chatbots
04752                 response. It's that simple. We call the chatbots
04753                 function to get the response by passing it the users
04754                 statement/question, etc.. The chatbot then searches the
04755                 database for a response which bests fits teh string
04756                 passed to the chatbots function. The returned data is
```

```
04757               then stored in the temp variable for later use and
04758               manipulation. This variable is used through out
04759               self.get_caprica_response() function except when
04760               the user has enabled the voice option and makes use
04761               of the voice option.
04762           self.master_log
04763               This is a string variable which as its name states
04764               contains a master log of the conversation. In other
04765               words, it stores both the users text and the chatbots
04766               text in order as it's entered. This is done so that we
04767               can write a full file of the entire conversation. As it
04768               occurs. This is not done real time. It's done when the
04769               user clicks the "Write logs button" which is represented
04770               by a pencil on the menu bar (Action Bar).
04771           self.username
04772               self.username contains the users username. This assumes
04773               that the user created a username. If the user did not
04774               create a username then a default value of 'User: '
04775               is provided. This is used in various ways: We set
04776               the view_port TextInput Widget conversation log
04777               with User: my statement. We append this data to the
04778               self.master_log string. We append this data to the
04779               User_statement.txt file.
04780           self.audio_disabled
04781               This is a boolean variable which we use to check whether
04782               or not the user has disabled the audio option. If
04783               self.audio_disabled == True; then the audio option is
04784               disabled and the chatbot can only communicate with the
04785               user via text. If self.audio_disabled == False then
04786               self.audio_enabled == True; meaning that the audio mode
04787               is enabled and the chatbot can access the systems
04788               text to speech software and communicate verbally with
04789               the user.
04790           self.audio_enabled
04791               This is a boolean variable which we use to check
04792               to see if the user has enabeld teh audio option.
04793               If the user has enabled the Audio option then Caprica
04794               can access the systems text to speech software and
04795               speak directly to the user.
04796               If self.audio_enabled == True then the user has
04797               enabled the Audio option; and the chatbot can then
04798               speak to the user verbally.
04799               If self.audio_enabled == False; then the user has either
04800               disabled the audio option or not bothered to enable it
04801               yet which means that the chatbot can only communicate
04802               with the usr via text.
04803           self.voice_enabled
04804               This is a boolean variable which we use to check
04805               to see if the user has enabled the voice option.
04806               The voice option enables the user to access and use
04807               their microphone to speak directly to Caprica. If
04808               the user doesn't have a microphone then they can't use
04809               this option. If self.voice_enabled == True the user has
04810               turned on the voice option and does have a microphone.
04811               If self.voice_disabled == False; the user has either
04812               disabled the voice option or doesn't have a microphone.
04813           temp
04814               The temp variable is used to store the chatbots
04815               response. It's that simple. We call the chatbots
04816               function to get the response by passing it the users
04817               statement/question, etc.. The chatbot then searches the
04818               database for a response which bests fits teh string
04819               passed to the chatbots function. The returned data is
04820               then stored in the temp variable for later use and
04821               manipulation. This variable is only used when the user
04822               has activated the voice option.
04823           self.mic
04824               self.mic is the initialized object of the
04825               SpeechRecognition.Microphone() class. With
04826               this object we can access the users microphone
04827               and listen to the audio then pass into the
04828               recognizer object for transcription; and later
04829               storage as a string.
04830           source
04831               The source variable is created to pipe the audio opened
04832               by self.mic; into the variable audio (which is an
04833               audio file). Source doesn't store the data. It simply
04834               passes the data into the audio variable as it's picked
04835               up by the users microphone. This of course assumes that
04836               the user has a microphone. If the user doesn't have a
04837               microphone then the user wont ever get into this
04838               function. The source object is cleared and destroyed
04839               when the with loop is ended. Using the with loop
04840               functionality automatically closes the loop, clears
04841               teh data, and deletes the object when the consditon
04842               reaches its breakpoint. In this case the breakpoint is
04843               when the user stops speaking. So basically, while this
```

```
04844                microphone is picking up noise pipe it through source
04845                and store it in the audio variable. When it stops picking
04846                up noise end the loop and clean up the data.
04847            statement
04848                This is a string variable which is used to store
04849                the string returned by the
04850                self.record.recognize_sphinx(audio) function. The
04851                audio file passed to the above mentioned function is
04852                transcribed and returned as a string to the statement
04853                variable.
04854        Members
04855        -------
04856            self.start_timer_thread(self, _time)
04857                This function is called when it's passed to the
04858                my_thread object. When the new thread is started
04859                this is used to call the self.caprica_timer()
04860                function. The double variable passed to it
04861                which represents the amount of time that
04862                self.caprica_timer() runs.
04863            self.chatbot..get_response(words)
04864                This function looks exeedingly complicated.
04865                But, it's not. Simply put this function is
04866                what takes the input from the user_input
04867                TextInput widget; passes it to the chatbot
04868                so it can locate an appropriate response by
04869                searching its database and then returns that
04870                string to either another variable or to a function.
04871                That string is then communicated to the user as
04872                the chatbots response. It checks to see if the
04873                user has enabled or disabled the audio and or
04874                voice modes. From there it accepts the input as
04875                it's intended to.
04876            threading.Thread(target = (), args = ())
04877                This creates a new thread, this thread
04878                refers to a function or other event
04879                and the parameters of arguements to be
04880                passed to that event. So in our case,
04881                we use this to start a timer which counts
04882                down from the number supplied to args; the
04883                function self.caprica_timer then preforms
04884                the count down which is checked by
04885                self.check_timer to ensure that the double
04886                variable stored in _time is less than
04887                or equal to zero. If it's equal to zero the
04888                function ends. If it's not equal to zero
04889                the function display the text '..Thinking..'
04890                in the notification_widget TextInput widget.
04891            threading.Thread().start()
04892                This function simply starts the new thread
04893                that was created. That is to say this function
04894                starts the my_thread thread.
04895            self.get_user_text()
04896                This function is used to return the text
04897                contained in self.ids.user_input.text in
04898                the form of a string.
04899            datetime.datetime.now().strftime()
04900                This function is called to return the current time
04901                in the form of a string. We use this to write the
04902                current time to a text file if an error occurs.
04903                This only executes if an error occurs.
04904            os.getlogin()
04905                This function is called to return the users
04906                system username. We use this function to create
04907                and manipulate text files. Think about it this way.
04908                On linux /home/user/folder is a filepath.
04909                The user portion of that refers to the users
04910                logged in username. Without the current users
04911                system username we can't write text files because we
04912                don't know the full path to any safe locations for us
04913                to write this data.
04914            sys.platform.startswith('platform')
04915                This function is called to check the user computers
04916                operating system. It checks a specific version number
04917                for each style of operating system. For instance,
04918                on windows this function checks registry keys and
04919                on linux it makes use of system call to return the
04920                major version string. On older linux systems this could
04921                return linux2, or linux3, or linux4, or linux1 etc.. In
04922                order to get around that we simply supply linux and parse
04923                the string to dertmine the version number. On windows
04924                it can return a variety of things such as win32.
04925                Supplying win as the parameter guarantees that we
04926                will determine if this is a windows based operating.
04927                We use this so that we can write the files and
04928                manipulate the program its self in a way that's
04929                compatible with the various operating systems.
04930            self.ids.user_input.text
```

```
04931                This function returns the string currently contained
04932                in the user_input TextInput widget. We use this
04933                to return the users string to the chatbot so that it
04934                can formulate an appropriate response for the user.
04935                As well as returning it to the appending of
04936                self.master_log, self.__append_file(dat, path) etc.
04937            self.ids.user_input.focus
04938                This function sets the current focus of the
04939                users mouse to user_input TextInput widget.
04940                Meaning that it's actively focused so the
04941                user doesn't have to click back into it.
04942                Unfortunately this is not having the desired
04943                effect on windows operating systems due to an
04944                ongoing issue with kivy and the windowing system
04945                on windows.
04946            self.caprica_speak(words)
04947                We call this function when the user has activated
04948                either the audio or voice options. We pass the
04949                chatbots generated response to this function.
04950                We then access the systems text to speech software
04951                to verbally "speak" the chatbots response to the
04952                user.
04953            self.ids.view_port.text
04954                We call this function to set the view_port TextInput
04955                widgets text property. We set this property to contain
04956                the users statement and the chatbots response in order.
04957                User: This is a statement.
04958                Caprica: Yes, that is a statement.
04959            self.record.listen(microphone_source)
04960                We call this function to open the users microphone
04961                assuming the user has a microphone. If they don'take
04962                have a microphone the user wont be able to access the
04963                voice option. If they do have a microphone this function
04964                turns the microphone on and enables it to accept noise.
04965                The noise is the users input, Ie, the users words. The
04966                microphone remains in an active state as long as the
04967                user speaks. That data is then piped into the recognizer
04968                for transciption into a string.
04969            self.recognize_sphinx(audio_file)
04970                This function is called after the user has made use
04971                of the voice option. It takes the audio file piped
04972                from the source variable to the audio variable which
04973                stores this data as an audio file. This audio file is
04974                then passed to self.recognize_sphinx(audio) which is
04975                then transcribed to a string and returned.
04976        Private Members
04977        ---------------
04978            self.__append_file(data, path)
04979                We call this function to write specific data to a
04980                specific text file. The text written to th files comes
04981                in two flavors. All of the chatbots response are written
04982                to the Caprica_Statements.txt file. All of the users
04983                statements are written to User_Statements.txt file.
04984                We use this to segregate the statements made by the user
04985                and the chatbot for later training purposes.
04986            self._stop_threading()
04987                This function is called to check to see if a thread is
04988                active. If a thread is active this function interupts
04989                the active thread which essentially (though not
04990                technically) kills it; to prevent memory leaks and
04991                a series of other potential issues.
04992        Exceptions
04993        ----------
04994            OSError
04995                The OSError can occur due to numerous reasons.
04996                What I'm primarily concerned with here however
04997                is import statements, incompatible Operating
04998                systems, and bad system calls. The exception
04999                if it occurs is handled and logged in an error
05000                log text file.
05001
05002            IOError
05003                The IOError can occur due to many reasons.
05004                My primary concern is file manipulation. The
05005                improper opening/closing/writing to files. If
05006                the exception occurs it's handled and logged; in
05007                an error log text file.
05008
05009            RunTimeError
05010                The RunTimeError error here is checking to make sure
05011                that the chat bot doesn't die. Essentially I just need
05012                to make sure that it completes and executes the python
05013                text to speech functions in a manner that doesn't cause
05014                a fatal exception. If something does occur the exception
05015                will be handled and logged to an error log text file.
05016
05017            ValueError
```

```
05018                 Ensures that values passed to the chat bot are
05019                 appropriate. And if for some reason one isn't the
05020                 exception will be handled and logged to an error
05021                 log text file.
05022         Returns
05023         -------
05024             None
05025         Notes
05026         -----
05027             So this is a large function and I explained it quite
05028             well broken down in the sections above. An overview of
05029             this function is this.
05030 
05031             We check to see if the user has enabled or disabled the
05032             audio and voice options. We then accept the users input
05033             in a way appropriate to the option the user has elected
05034             to use. We then obtain a response from the chatbot and
05035             either send it to the user in text or audio form.
05036         '''
05037         try:
05038             if sys.platform.startswith('linux'):
05039                 if self.audio_disabled:
05040                     response = str(self.chatbot.get_response(self.
      get_user_text()))
05041                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + self.get_user_text(), '/home/' + str(os.getlogin()) + '
      /.SentienceFiles/User_Statements.txt')
05042                     self.master_log += '\n' + self.username + ': ' + self.
      get_user_text()
05043                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + response, '/home/' + str(os.getlogin()) + '/.SentienceFiles/Caprica_Statements.txt')
05044                     self.master_log += '\nCaprica: ' + response
05045                     self.ids.view_port.text = self.username + ': ' + self.
      get_user_text() + '\nCaprica: ' + response
05046                     self.ids.user_input.text = ''
05047                     self.ids.user_input.focus = True
05048                     time.sleep(1)
05049                     self.__currently_thinking(False)
05050                 elif self.audio_enabled:
05051                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + self.get_user_text(), '/home/' + str(os.getlogin()) + '
      /.SentienceFiles/User_Statements.txt')
05052                     self.master_log += '\n' + self.username + ': ' + self.
      get_user_text()
05053                     response = str(self.chatbot.get_response(self.
      get_user_text()))
05054                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + response, '/home/' + str(os.getlogin()) + '/.SentienceFiles/Caprica_Statements.txt')
05055                     self.master_log += '\nCaprica: ' + response
05056                     self.ids.view_port.text = self.username + ': ' + self.
      get_user_text() + '\nCaprica: ' + response
05057                     self.caprica_speak(response)
05058                     self.ids.user_input.focus = True
05059                     time.sleep(1)
05060                     self.__currently_thinking(False)
05061                 elif self.voice_disabled:
05062                     self.ids.view_port.text = 'Please activate the voice option by clicking on the red
       microphone button'
05063                     return None
05064                 elif self.voice_enabled:
05065                     with self.mic as source:
05066                         audio = self.record.listen(source)
05067                     statement = self.record.recognize_sphinx(audio)
05068                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + statement, '/home/' + str(os.getlogin()) + '/.SentienceFiles/User_Statements.txt')
05069                     self.master_log += '\n' + self.username + ': ' + str(statement)
05070                     temp = str(self.chatbot.get_response(statement))
05071                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp, '/home/' + str(os.getlogin()) + '/.SentienceFiles/Caprica_Statements.txt')
05072                     self.master_log += '\nCaprica: ' + temp
05073                     self.ids.view_port.text = self.username + ': ' + str(statement) + '\nCaprica: '
       + str(temp)
05074                     self.caprica_speak(temp)
05075                     time.sleep(1)
05076                     self.__currently_thinking(False)
05077             elif sys.platform.startswith('win'):
05078                 if self.audio_disabled:
05079                     response = self.chatbot.get_response(self.
      get_user_text())
05080                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + ' ' + self.get_user_text(),"C://SentienceFiles//User_Statements.txt")
05081                     self.master_log += '\n' + self.username + ': ' + self.
      get_user_text()
05082                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + ' ' + str(response),"C://SentienceFiles//Caprica_Statements.txt")
05083                     self.master_log += '\nCaprica: ' + str(response)
05084                     self.ids.view_port.text = self.username + ': ' + self.
```

```
get_user_text() + '\nCaprica: ' + str(response)
05085                     self.ids.user_input.text = ''
05086                     self.ids.user_input.focus = True
05087                     time.sleep(1)
05088                     self.__currently_thinking(False)
05089                 elif self.audio_enabled:
05090                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
 %H:%M:%S')) + self.get_user_text(), 'C://SentienceFiles//User_Statements.txt')
05091                     self.master_log += '\n' + self.username + ': ' + self.
get_user_text()
05092                     response = self.chatbot.get_response(self.
get_user_text())
05093                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
 %H:%M:%S')) + str(response), 'C://SentienceFiles//Caprica_Statements.txt')
05094                     self.master_log += '\nCaprica: ' + str(response)
05095                     self.ids.view_port.text = self.username + ': ' + self.
get_user_text() + '\nCaprica: ' + str(response)
05096                     self.caprica_speak(str(response))
05097                     self.ids.user_input.focus = True
05098                     time.sleep(1)
05099                     self.__currently_thinking(False)
05100                 elif self.voice_disabled:
05101                     self.ids.view_port.text = 'Please activate the voice option by clicking on the red
 microphone button'
05102                     return None
05103                 elif self.voice_enabled:
05104                     with self.mic as source:
05105                         audio = self.record.listen(source)
05106                     statement = self.record.recognize_sphinx(audio)
05107                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
 %H:%M:%S')) + statement, 'C://SentienceFiles//User_Statements.txt')
05108                     self.master_log += '\n' + self.username + ': ' + statement
05109                     temp = self.chatbot.get_response(statement)
05110                     self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
 %H:%M:%S')) + str(temp), 'C://SentienceFiles//Caprica_Statements.txt')
05111                     self.master_log += '\nCaprica: ' + str(temp)
05112                     self.ids.view_port.text = self.username + ': ' + statement + '\nCaprica: ' +
str(temp)
05113                     self.caprica_speak(str(temp))
05114                     time.sleep(1)
05115                     self.__currently_thinking(False)
05116         except OSError as c:
05117             if sys.platform.startswith('linux'):
05118                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nOSError: '
 + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05119             elif sys.platform.startswith('win'):
05120                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nOSError: '
 + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
C://SentienceFiles//Error Logs.txt')
05121             return None
05122         except IOError as d:
05123             if sys.platform.startswith('linux'):
05124                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nIOError: '
 + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05125             elif sys.platform.startswith('win'):
05126                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\nIOError: '
 + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
C://SentienceFiles//Error Logs.txt')
05127             return None
05128         except RuntimeError as e:
05129             if sys.platform.startswith('linux'):
05130                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
RuntimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05131             elif sys.platform.startswith('win'):
05132                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
RuntimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
C://SentienceFiles//Error Logs.txt')
05133             return None
05134         except ValueError as f:
05135             if sys.platform.startswith('linux'):
05136                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str
(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05137             elif sys.platform.startswith('win'):
05138                 self.__append_file('\n' + 'Function: get_user_voice_response ' + '\n
ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
C://SentienceFiles//Error Logs.txt')
05139             return None
05140
05141
05142
05143     def get_caprica_voice_thread(self, words):
05144         '''
05145         get_caprica_voice_thread(self, words)
```

```
05146
05147         Parameters
05148         ----------
05149             param1 : self
05150                 Denotes this as being a member of SentienceScreen().
05151             param2 : words
05152                 A string containing the users transcribed voice response
05153                 is passed to this for later manipulation.
05154         Attributes
05155         ----------
05156             temp
05157                 temp is a string variable that is used to temporarily
05158                 store the generated response of the chatbot. This
05159                 variable will then be written to varios files and
05160                 displayed in the view_port TextInput widget.
05161             self.master_log
05162                 self.master_log is a string variable wich contains a
05163                 master conversation log. This log includes the text
05164                 sent by the user and the responses generated by
05165                 the chatbot as they appear.
05166             self.username
05167                 self.username is a string variable which contains the
05168                 users chosen username. IF the user did not elect to
05169                 setup a user profile a default value of 'User: ' is
05170                 provided.
05171         Members
05172         -------
05173             self.caprica_speak(words)
05174                 This function is called after the user has sent
05175                 text to the chatbot. That text is then passed to
05176                 this funtction if self.audio_enabled == True
05177                 or if self.voice_enabled == True. We then
05178                 access the users systems text to speech software
05179                 to verbally speak the passed string. This function
05180                 is a member of SentienceScreen().
05181             time.sleep(integer)
05182                 We call time.sleep(1) to force the program to
05183                 sleep for one second. This ensures that certain
05184                 functions are called by forcing the Kivy.clock() to
05185                 appropriately execute events in the correct order
05186                 in the frame. It also prevents the program from hanging
05187                 by trying to execute things to fast. This function is a
05188                 member of the class time.
05189             sys.platform.startswith(string)
05190                 We call this function to determine the users operating
05191                 system. If the user is running a windows system then
05192                 the appropriate if statements execute. If they're
05193                 running a linux system again the appropriate if
05194                 statements are executed. We determine this by accessing
05195                 the systems major version. For instance, older
05196                 linux systems return values such as, linux1, linux2,
05197                 linux3 and so on. Windows systems may return win32 etc.
05198                 By checking the preceeding version string, Ie, 'linux'
05199                 we know it's a linux system, or 'win' we know it's a
05200                 windows system. Where these strings comes from varies
05201                 based on the operating system. On linux it's an os call.
05202                 On windows it's a registry key. This function is a
05203                 member of the class sys.
05204             datetime.datetime.now().strftime(string)
05205                 We call this function to return the current local time.
05206                 We format it to ourput in
05207                 year, month, day, hours, minutes seconds. This is
05208                 returned as a string directly to our write method.
05209             os.getlogin()
05210                 This function is a member of the os class. We call
05211                 this function to return the current users system user
05212                 name. We do this so that we can succesfully write files
05213                 to the users system. On linux the filesystem has the
05214                 users name as part of it's non root path. We need this
05215                 name to access the location where we want to write to.
05216         Private Members
05217         ---------------
05218             self.__append_file(string, path)
05219                 This function is a member of SentienceScreen(). We
05220                 call this function to append specific data to
05221                 specific text files. The data is passed in as a string.
05222                 As is the path to the file.
05223             self._stop_threading()
05224                 This function is a member of the SentienceScreen()
05225                 class. we call this function to interupt our running
05226                 threads.
05227             self.__currently_thinking(boolean)
05228                 This function is a member of the SentienceScreen()
05229                 class. We call this function to change the current
05230                 banner which informs the user if the program is
05231                 currently inactive or thinking. It change the text to
05232                 either '...Thinking...' with a red foreground. Or,
```

```
05233                 '...Inactive...' with a blue foreground. The boolean
05234                 variable dictates whether or not the program is in fact
05235                 actively thinking or not.
05236         Exceptions
05237         ----------
05238              OSError
05239                  The OSError can occur due to numerous reasons.
05240                  What I'm primarily concerned with here however
05241                  is import statements, incompatible Operating
05242                  systems, and bad system calls. The exception
05243                  if it occurs is handled and logged in an error
05244                  log text file.
05245
05246              IOError
05247                  The IOError can occur due to many reasons.
05248                  My primary concern is file manipulation. The
05249                  improper opening/closing/writing to files. If
05250                  the exception occurs it's handled and logged; in
05251                  an error log text file.
05252
05253              RunTimeError
05254                  The RunTimeError error here is checking to make sure
05255                  that the chat bot doesn't die. Essentially I just need
05256                  to make sure that it completes and executes the python
05257                  text to speech functions in a manner that doesn't cause
05258                  a fatal exception. If something does occur the
05259                  exception will be handled and logged to an error log
05260                  text file.
05261
05262              ValueError
05263                  Ensures that values passed to the chat bot are
05264                  appropriate. And if for some reason one isn't the
05265                  exception will be handled and logged to an error
05266                  log text file.
05267         Returns
05268         -------
05269             None
05270         Notes
05271         -----
05272             This function is one our response threads. We call it
05273             if the user has activated the voice feature. Ie, if
05274             self.voice_enabled == True, this function will be called
05275             when the user clicks on the enable/disable voice button
05276             which is represented by a red or blue microphone on the
05277             menu bar (Action Bar).
05278
05279             We determine the users operating system. Obtain the chatbots
05280             generated response. Next we append it to
05281             the Caprica_Statements text file. We then save it to the
05282             master log. We then display the response and the users
05283             initial text in the view_port TextInput widget. Next we
05284             call self.caprica_speak(words) to actually verbally
05285             communicate the chat bots response to the user.
05286
05287             We next interupt the thread and put the program to sleep
05288             for one second by calling time.sleep(1). We finally
05289             call self.__currently_thinking(False) to reset the
05290             banner text to '...Thinking...' with a blue foreground.
05291
05292         '''
05293         try:
05294             if sys.platform.startswith('linux'):
05295                 temp = str(self.chatbot.get_response(words))
05296                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + temp, '/home/' + str(os.getlogin()) + '/.SentienceFiles/Caprica_Statements.txt')
05297                 self.master_log += '\nCaprica: ' + temp
05298                 self.ids.view_port.text = self.username + ': ' + str(words) + '\nCaprica: ' + str(
      temp)
05299                 self.caprica_speak(temp)
05300                 time.sleep(1)
05301                 self.__currently_thinking(False)
05302             elif sys.platform.startswith('win'):
05303                 temp = str(self.chatbot.get_response(words))
05304                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
       %H:%M:%S')) + str(temp), 'C://SentienceFiles//Caprica_Statements.txt')
05305                 self.master_log += '\nCaprica: ' + str(temp)
05306                 self.ids.view_port.text = self.username + ': ' + str(words) + '\nCaprica: ' + str(
      temp)
05307                 self.caprica_speak(str(temp))
05308                 time.sleep(1)
05309                 self.__currently_thinking(False)
05310         except OSError as c:
05311             if sys.platform.startswith('linux'):
05312                 self.__append_file('\n' + 'Function: get_caprica_voice_thread ' + '\nOSError:
      ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05313             elif sys.platform.startswith('win'):
```

```
05314                 self.__append_file('\n' + 'Function: get_caprica_voice_thread ' + '\nOSError:
      ' + str(c) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
05315             return None
05316         except IOError as d:
05317             if sys.platform.startswith('linux'):
05318                 self.__append_file('\n' + 'Function: get_caprica_voice_thread  ' + '\nIOError:
       ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05319             elif sys.platform.startswith('win'):
05320                 self.__append_file('\n' + 'Function: get_caprica_voice_thread  ' + '\nIOError:
       ' + str(d) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
05321             return None
05322         except RuntimeError as e:
05323             if sys.platform.startswith('linux'):
05324                 self.__append_file('\n' + 'Function: get_caprica_voice_thread  ' + '\n
      RuntimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
       str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05325             elif sys.platform.startswith('win'):
05326                 self.__append_file('\n' + 'Function: get_caprica_voice_thread  ' + '\n
      RuntimeError: ' + str(e) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
05327             return None
05328         except ValueError as f:
05329             if sys.platform.startswith('linux'):
05330                 self.__append_file('\n' + 'Function: get_caprica_voice_thread  ' + '\n
      ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' +
      str(os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05331             elif sys.platform.startswith('win'):
05332                 self.__append_file('\n' + 'Function: get_caprica_voice_thread  ' + '\n
      ValueError: ' + str(f) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
05333             return None
05334
05335
05336
05337     def start_get_response_thread(self):
05338         '''
05339         self.start_get_response_thread(self)
05340
05341         Parameters
05342         ----------
05343             param1 : self
05344
05345         Attributes
05346         ----------
05347             target
05348                 target is a member of the Process() class.
05349         Members
05350         -------
05351             self.ids.notification_widget.text
05352                This is a member of SentienceScreen() and is the
05353                notification_widget TextInput Widget in the kv
05354                design language. We use this to set the current text of the
05355                notification_widget TextInput Widget to the supplied string.
05356                This widget is used to display the text '..Thinking..'
05357                while the chatbot is locating an appropriate response for
05358                the user.
05359             self.ids.notification_widget.foreground_color
05360                 This is a member of SentienceScreen() and is the
05361                 notification_widget TextInput Widget in the kv
05362                 design language. We use this to set the current
05363                 text color of the widget. Here we see two examples.
05364                 If we change the text to ...Thinking... we change the
05365                 color to red; this shows the user that the program is
05366                 active and that the chatbot is currently generating a
05367                 response. If we change the text to ...Inactive... then
05368                 we change the text color to blue. This tells the user
05369                 that the chatbot has generated a response and that
05370                 the user can once again speak to it.
05371             Threading.Thread(target = event, args = (tuple)).start()
05372                 threading.Thread() is called to create a new thread.
05373                 This is used to run the acquisition of the a response
05374                 from the chatbot to the user. We pass the
05375                 function self.get_caprica_response to this as its event
05376                 and because self.get_caprica_response doesn't require
05377                 any arguements we omit the "args" parameter that we
05378                 saw in the threading code. We then call start() to
05379                 actually run the thread.
05380             Time.sleep(time_interval)
05381                 We call this function to force the program to "sleep",
05382                 for 1 second. This executes the two lines above
05383                 Time.sleep(1) before it creates and starts the thread.
05384                 We do this to prevent hanging issues caused by the
05385                 threading event and to ensure that the proper text
05386                 and color of the text is set; and more importantly,
```

```
05387                so that the user can see this text.
05388            kivy.utils.get_color_from_hex('Hex string')
05389                This is a kivy function. It's a member of kivy.utilsself.
05390                We call this function to convert a hexadecimal string
05391                into an equivelant opengl based rbga color.
05392            sys.platform.os.startswith('string')
05393                This is a member of the Python sys (system) library.
05394                We call this function to check the major version
05395                of the users operating system. By doing so we can
05396                determine if the user is running a windows or linux based
05397                operating system. Note: See how I said "A" instead of
05398                windows 10 or ubuntu? By using .startswith('') we
05399                simply detect the operating system and are able
05400                to be truly cross platform.
05401            time.sleep()
05402                This is a member of the Time class(). We call this
05403                function to force the program to sleep for one second.
05404                To ensure that the notification_widget TextInput
05405                text property reflects the current state of the program.
05406        Private Members
05407        ---------------
05408            self.__set_thinking_text
05409                We call this function to inform the user that the
05410                chatbot is about to generate a response. The
05411                self.notification_widget has its text set to
05412                '...Thinking...' and the text is also made red.
05413        Exceptions
05414        ----------
05415            None
05416        Returns
05417        -------
05418            None
05419        Notes
05420        -----
05421            This function is called when the user sends a response to
05422            the chatbot. We use this to start self.get_caprica_response
05423            as a new thread to improve performance.
05424
05425            We first detect the operating system, then set the text
05426            of self.notification_widget to be appropriate, we then
05427            change the color to reflect the text as mentioned
05428            above. We then force the program to sleep for one second
05429            to ensure those changes take effect. We finally create and
05430            run a new thread which then starts the
05431            self.get_caprica_response() function.
05432        '''
05433        if sys.platform.startswith('linux'):
05434            if len(self.get_user_text()) <= 1:
05435                self.ids.view_port.text = 'Please enter at least two characters. Like Hi'
05436                self.ids.user_input.text = ''
05437            elif len(self.get_user_text()) > 1:
05438                self.__currently_thinking(True)
05439                time.sleep(1)
05440                threading.Thread(name = 'linux_thread', target = self.
    get_caprica_response).start()
05441        elif sys.platform.startswith('win'):
05442            if len(self.get_user_text()) <= 1:
05443                self.ids.view_port.text = 'Please enter at least two characters. Like Hi'
05444                self.ids.user_input.text = ''
05445            elif len(self.get_user_text()) > 1:
05446                self.__currently_thinking(True)
05447                time.sleep(1)
05448                threading.Thread(name = 'windows_thread', target = self.
    get_caprica_response).start()
05449
05450
05451
05452    def start_voice_response_thread(self):
05453        '''
05454        start_voice_response_thread(self)
05455
05456        Parameters
05457        ----------
05458            param1 : self
05459                Denotes this as being a member of the SentienceScreen()
05460                class.
05461        Attributes
05462        ----------
05463            self.voice_disabled
05464                This is a member of the SentienceScreen() class.
05465                We use this boolean variable asa flag o tell us
05466                whether or not the user enabled or disabled the
05467                voice option. If the user has disabled the voice
05468                they will informed that the voice option is
05469                disabled and that they need to enable it. They
05470                can do so by clicking on the red microphone
05471                button on the menu bar (Action Bar).
```

```
05472             self.vocice_enabled
05473                 This is a member of the SentienceScreen() class. We
05474                 we use this boolean variable as flag to tel us
05475                 whether or not the user has enabled or disabled
05476                 the voice option. If the voice option is activated
05477                 the users microphone will be opened and voice input
05478                 will be recorded as long as the microphone picks up
05479                 noise.
05480             self.mic
05481                 self.mic is a member of the SentienceScreen() class.
05482                 It's also the instantiated object of sr.Microphone).
05483                 We use this object to open the users microphone if
05484                 they have one and pipe the input through sthe source
05485                 variable. When the user stops speaking the microphone
05486                 should close the audio in source which is being
05487                 listened to is then returned to the audio variable.
05488             source
05489                 source is a local variable of the function
05490                 self.get_caprica_voice_thread(). We use this variable
05491                 to store the input piped from the users microphone.
05492                 Once the microphone stops picking up audio input
05493                 we then transcribe the audio data into a string
05494                 by passing it to self.recognize_sphinx(audio).
05495             statement
05496                 statement is a local variable of the function
05497                 self.get_caprica_voice_thread() we use it to
05498                 store the transcribed string which is returned
05499                 to us by the self.recognize_sphinx(audio) function.
05500             audio
05501                 audio is a local variable of the function
05502                 self.get_caprica_voice_thread(). We use this
05503                 variable to store the audio data collected
05504                 by source which was piped through the users
05505                 microphone. We then pass this variable to
05506                 self.recognize_sphinx(audio). Which is then
05507                 transcribed from audio data and returned as
05508                 a string and stored in the statement variable.
05509             self.master_log
05510                 self.master_log is a member of the SentienceScreen()
05511                 class. This variable is used to store the full
05512                 conversation between the chatbot and the user. This
05513                 variable is later used to write data to a file.
05514         Members
05515         -------
05516             sys.platform.startswith(string)
05517                 This is a member of the sys() class. We call this
05518                 function to find out which operating system the
05519                 user is running. To be specific, we're only checking
05520                 for windows and linux based operating systems. This
05521                 funtion returns True if it matches 'linux', if
05522                 this happens we know that the user is running a
05523                 linux based operating system. If it returns False,
05524                 we then check to see if the user is running a windows
05525                 based operating sytem by passing 'win' to the function.
05526             self.record.listen(source)
05527                 We call this function to "listen" or, accept and store
05528                 the audio being piped through the users microphone
05529                 into the source variable. When the microphone no
05530                 longer detects audio input this audio data is
05531                 returned and stored in the audio variable.
05532             self.ids.view_port.text
05533                 This is a member of the SentienceScreen() class. We use
05534                 this to set the view_port TextInput widgets text field.
05535                 If the user hasn't enabled the voice option we
05536                 inform the user that the voice option is currently
05537                 disabled and that they can enable it by clicking
05538                 on the red mirophone button on the menu bar (Action
05539                 Bar).
05540             self.record.recognize_sphinx(audio)
05541                 We call this function to transcribed the passed
05542                 audio file into a string. The audio passed was
05543                 collected via the users microphone. This audio
05544                 data is transcribed into a string and returned and
05545                 stored in the statement variable.
05546             threading.Thread(*args)
05547                 This is a member of the threading() class. We use
05548                 this to declare, instantiate and run our thread all at
05549                 once. The thread is given a name based on the users
05550                 operating system. Ie, if it's linux, it's named
05551                 'linux_thread' and if it's windows it's named
05552                 'windows_thread'. We then pass it the target event
05553                 which is self.get_caprica_voice_thread(statement).
05554                 We finally call the start() function of the
05555                 threading class to actually start the new thread.
05556             datetime.datetime.now().strftime(string)
05557                 This is a member of the datetime() class.
05558                 We use this function to return the current local time
```

```
05559                 inside our file appending function. The time format
05560                 is set to year, month, day, hour, minute, seconds.
05561             self.get_caprica_voice_thread(string)
05562                 This is a member of the SentienceScreen() class.
05563                 We call this function as the event which is the
05564                 thread. That is to say this is the new thread. We
05565                 pass it the users transcribed verbal statement which
05566                 is stored in the string variable statement.
05567             time.sleep(integer)
05568                 This is a member of the time() class. We call
05569                 this function to put the program to sleep for one
05570                 second. We do this to ensure that the thread is
05571                 not executed until after the text, and the color of
05572                 the text in the self.notification_widget TextInput
05573                 widget have been changed to reflect the programs
05574                 current status.
05575         Private Members
05576         ---------------
05577             self.__append_file(string, path)
05578                 This is a member of the SentienceScreen() class.
05579                 We call this function and pass it the string
05580                 which contains the users response to the chatbot
05581                 as well as the date and time that this response
05582                 occured. We then supply it with the absolute file
05583                 path of the file that we're writing to which is
05584                 the User_Statements.text file. This path depends on
05585                 the users operating system.
05586             self.__currently_thinking(bool)
05587                 This is a member of the SentienceScreen() class.
05588                 We call this function to set the current text and
05589                 color of that text to reflect the programs current
05590                 status. We pass it a boolean variable which is used
05591                 to determine this status. For more information on
05592                 this function see its comments.
05593         Exceptions
05594         ----------
05595             None
05596         Returns
05597         -------
05598             None
05599         Notes
05600         -----
05601             This function is really straightforward. We use this
05602             when the user clicks on the record user button. Which is
05603             represented by the blue talking human head on the menu bar.
05604
05605             If the button is red when the user clicks it, it means that
05606             the voice option wasn't enabled. We then inform the user
05607             in the view_port TextInput widget that the voice option
05608             is not currently enabled. We also inform them how to
05609             activate this option.
05610
05611             Once the voice mode is activated and the record user
05612             button has been clicked. We open the users mirophone
05613             and pipe the audio input into source which is passed
05614             to the listen() function and stored in its audio form.
05615
05616             When the microphone stops picking up audio input
05617             listen() function terminates and returns the audio
05618             to the local variable named audio.
05619
05620             We then pass the audio variable into the function
05621             recognize_sphinx() which transcribes it into a string.
05622             Returns that string to be stored in the local variable
05623             named statement.
05624
05625             We then append the users response to the chatbot as well
05626             as the date and time this response occured to the
05627             User_Statement text file. We then add this response to
05628             the end of the master_log string. Along with the users
05629             username.
05630
05631             We then set the current status of the chatbot, Ie,
05632             thinking or inactive. Which is then reflected in
05633             the notification_widget text property. We then call
05634             time.sleep() and give it a one second interval to
05635             ensure that the above does occur before the thread
05636             is setup and run.
05637         '''
05638         if sys.platform.startswith('linux'):
05639             if self.voice_disabled:
05640                 self.ids.view_port.text = 'Please activate the voice option by clicking on the red
      microphone button'
05641                 return None
05642             elif self.voice_enabled:
05643                 with self.mic as source:
05644                     audio = self.record.listen(source)
```

```
05645                 statement = self.record.recognize_sphinx(audio)
05646                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
      %H:%M:%S')) + statement, '/home/' + str(os.getlogin()) + '/.SentienceFiles/User_Statements.txt')
05647                 self.master_log += '\n' + self.username + ': ' + str(statement)
05648                 self.__currently_thinking(True)
05649                 time.sleep(1)
05650                 threading.Thread(name = 'linux_thread', target = self.
     get_caprica_voice_thread(statement)).start()
05651         elif sys.platform.startswith('win'):
05652             if self.voice_disabled:
05653                 self.ids.view_port.text = 'Please activate the voice option by clicking on the red
      microphone button'
05654                 return None
05655             elif self.voice_enabled:
05656                 with self.mic as source:
05657                     audio = self.record.listen(source)
05658                 statement = self.record.recognize_sphinx(audio)
05659                 self.__append_file('\n' + str(datetime.datetime.now().strftime('%Y-%m-%d
      %H:%M:%S')) + str(statement), 'C://SentienceFiles//User_Statements.txt')
05660                 self.master_log += '\n' + self.username + ': ' + str(statement)
05661                 self.__currently_thinking(True)
05662                 time.sleep(1)
05663                 threading.Thread(name = 'windows_thread', target = self.
     get_caprica_voice_thread(statement)).start()
05664
05665
05666
05667     def get_user_text(self):
05668         '''
05669         We call this function to return the text contained in
05670         self.ids.user_input.text TextInput Widget; in the form of a
05671         string.
05672         '''
05673         return str(self.ids.user_input.text)
05674
05675
05676
05677     def open_delete_file_dialog(self):
05678         '''
05679         open_delete_file_dialog(self)
05680
05681         Parameters
05682         ----------
05683             param1 : self
05684                 Denots this as being a member of SentienceScreen().
05685         Attributes
05686         ----------
05687             content
05688                 content is local variable to the function
05689                 self.open_delete_file_dialog() we use this
05690                 to make a new Object. This object is DeleteDialog
05691                 we then add the instantiated object to Popup().
05692                 Note: We add the local object content which is
05693                 the DeleteDialog to the kivy Popup() content field.
05694                 We do this because of how the layouts work.
05695             delete_file
05696                 delete_file is the ObjectProperty() that we
05697                 created in the DeleteDialog() class. We're using
05698                 this property, to bind it to the
05699                 SentienceScreen.().delete_file() function.
05700             self.popup
05701                 self._popup is the decleration and initialization
05702                 of the kivy Popup(). We've created this object, this
05703                 Popup() window and can now call it at anytime.
05704             Cancel
05705                 Cancel is the ObjectProperty() that we
05706                 created in the DeleteDialog() class. We're using
05707                 this property, to bind it to the
05708                 SentienceScreen.().dismiss_popup() function.
05709             title
05710                 title is a member of the Kivy Popup() class. It's
05711                 a StringProperty() which we use to set the title of
05712                 the Popup() window.
05713             size_hint
05714                 size_hint is member of the Kivy Popup() class, and
05715                 all other kivy widgets. We use size_hint to to set
05716                 the size of the widget. This enables us to set a
05717                 size based of percentages of the users monitor size.
05718                 Simply put, this enables us to create a size that's
05719                 compatiable with all devices and will stretch and
05720                 shrink in a manner that wont distort the programs
05721                 appearance.
05722         Members
05723         -------
05724             Popup()
05725                 Popup() is a kivy widget which is exactly what it
05726                 sounds like. It's a popup window. It's not a new window,
```

```
05727                it's a widget wich locks to the MainWindow (root window);
05728                we use this to allow the user to navigate to a
05729                specific file, select that file, and then delete it.
05730            DeleteDialog(FloatLayout)
05731                DeleteDialog is a class with a default FloatLayout
05732                that we created earlier. We use this class and its
05733                layout with our Popup() widget.
05734            self._popup.open()
05735                We call this function to add the Popup() widget
05736                to the screen.
05737            self.delete_file(self, path, filename)
05738                self.delete_file(self, path, filename) is the
05739                same function that we bound to the delete_file
05740                ObjectProperty; we just ommited its parameters
05741                when we did it. It recieves its arguements when
05742                the users selects a file (clicks on one) in the
05743                Popup() window and then clicks the delete button
05744                in the Popup() window. This function is passed
05745                the name of the file and it's file path. I
05746                actually don't use all of the parameters. But
05747                both of them could be used.
05748            self.dismiss_popup()
05749                We call this function to delete the Popup() window
05750                from the screen.
05751        Private members
05752        ---------------
05753            None
05754         Returns
05755         -------
05756             None
05757         Exceptions
05758         ----------
05759             None
05760         Notes
05761         -----
05762             This function is pretty straight forward. When the user
05763             clicks on the "Delete File" button a popup window is
05764             created and then added to the screen. This window contains
05765             a file browser, and two buttons. The file browser allows
05766             the user to navigate through their file system and select
05767             a file that they wish to delete.
05768
05769             Once the user has located the file they wish to delete
05770             they simply click on that file, which selects it, and
05771             then click the "Delete" button in the popup window.
05772
05773             This then returns the filename and path of the file to
05774             the self.delete_file(self, path, file). Which then preforms
05775             the deletion operation.
05776
05777             We then finally close the popup by removing (deleting) it
05778             from the MainWindow.
05779         '''
05780         content = DeleteDialog(delete_file = self.delete_file, Cancel = self.
     dismiss_popup)
05781         self._popup = Popup(title = "Select File For Deletion", content = content, size_hint = (0.9,
     0.9))
05782         self._popup.open()
05783
05784
05785
05786     def delete_file(self, path, filename):
05787         '''
05788         delete_file(self, path, filename)
05789
05790         Parameters
05791         ----------
05792             param1 : self
05793                 Denotes this as being a member of the SentienceScreen()
05794                 class.
05795
05796             param2 : path
05797                 contains the partial path to the file
05798                 returned to it by the users selection in the
05799                 DeleteDialog() pop up window. This parameter
05800                 does not contain the filename. Nor is it ever
05801                 used. It's pointless to even be here.
05802
05803             param3: filename
05804                 filename contains the full file path to the
05805                 file that the user selected for deletion by
05806                 clicking on the file and then clicking the
05807                 delete button in the DeleteDialog() Popup()
05808                 window.
05809         Attributes
05810         ----------
05811             temp
```

```
05812                 temp contains teh formatted and fbsolute filepath
05813                 to the file that the user selected for deletion. This
05814                 path is then passed to os.remove(temp) to carry out
05815                 the actual deletion process.
05816         Members
05817         -------
05818             os.path.isfile(path)
05819                 We call this function which is a member of the
05820                 os() class. To ensure that the filepath passed to
05821                 it does indeed exist. If it does exist this function
05822                 returns tru and the appropriate if statement is
05823                 executed. If it returns False the nthe file does not
05824                 exist and again the appropriate if statement is
05825                 executed.
05826
05827         os.remove(path)
05828             We call this function which is a member of the os()
05829             class to carry out the deletion process of the file
05830             that the user selected. This call makes use of the
05831             users systems native API deletion feature.
05832
05833         self.ids.view_port.text
05834             We call this function which is a member of the
05835             SentienceScreen() class. To change the text of the
05836             view_port TextInput widget. The text set depends on
05837             whether or not the file was deleted. If the file
05838             selected did exist and was deleted the text is
05839             changed to 'Filepath File has been deleted'. If
05840             the file did not exist and therefor was not deleted
05841             then the text is set to 'Filepath either does not exist
05842             or was already deleted'.
05843         Private Members
05844         ---------------
05845             None
05846         Exceptions
05847         ----------
05848             IOError
05849                 The IOError can occur due to many reasons.
05850                 My primary concern is file manipulation. The
05851                 improper opening/closing/writing to files. If
05852                 the exception occurs it's handled and logged; in
05853                 an error log text file.
05854
05855             FileNotFoundError
05856                 This can occur in a variety of ways however my primary
05857                 concern is that file path the user selected is broken.
05858                 Resulting in an File Not Found error. If this occurs
05859                 it's handled and logged to an error file text log.
05860         Returns
05861         -------
05862             None
05863         Notes
05864         -----
05865             This function is called after the user selects a file
05866             that they wish to delete in the DeleteDialog() Popup()
05867             window. Then proceeds by clicking the delete button on
05868             that Popup() window. The partial path with out the
05869             file name is returned to this funtion but is not used.
05870
05871             The full filepath is also returned and stored in the
05872             variable filename. We store filename in the string variable
05873             temp. We then strip the first two, and last two characters
05874             in the temp variable. Filename is returned as tupple and
05875             so it contains '[/example/filepath/random.text]'.
05876
05877             We then ensure that the selected file does exist
05878             and if it does we delete it and inform the user
05879             that the file was succesfully deleted. If the file
05880             doesn't exist during the initial check we then
05881             inform the user that it doesn't exist or that
05882             it's already been deleted.
05883         '''
05884         try:
05885             temp = str(filename)
05886             temp = temp[2:-2]
05887             if os.path.isfile(temp):
05888                 print(temp + ' Has been deleted')
05889                 os.remove(temp)
05890                 self.ids.view_port.text = str(temp) + ' File has been deleted.'
05891             elif not os.path.isfile(temp):
05892                 print(temp + ' Either does not exist or was already deleted.')
05893         except IOError as a:
05894             if sys.platform.startswith('linux'):
05895                 self.__append_file('\n' + 'Function: delet_files ' + '\nIOError: ' + str(a) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin())
       + '/.SentienceFiles/Error Logs.txt')
05896             elif sys.platform.startswith('win'):
```

```
05897                 self.__append_file('\n' + 'Function: delete_files ' + '\nIOError: ' + str(a) +
      '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
      Logs.txt')
05898             return None
05899         except FileNotFoundError as b:
05900             if sys.platform.startswith('linux'):
05901                 self.__append_file('\n' + 'Function: delet_files ' + '\nFileNotFoundError: ' +
      str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
     os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
05902             elif sys.platform.startswith('win'):
05903                 self.__append_file('\n' + 'Function: delete_files ' + '\nFileNotFoundError: '
     + str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
     C://SentienceFiles//Error Logs.txt')
05904             return None
05905
05906
05907
05908     def delete_all(self):
05909         '''
05910         delete_all(self)
05911
05912         Paraemters
05913         ----------
05914             param1 : self
05915                 Denotes this as being a member of
05916                 SentienceScreen().
05917         Attributes
05918         ----------
05919             temp
05920                 temp is a local variable which we use to store the
05921                 absolute directory path; to the folder that was
05922                 created when this program was first run. This folder
05923                 varies based on the users operating system. We pass
05924                 this variable to our deletion tool.
05925          ignore_errors
05926              This is sort of a misnomer. This a member of the
05927              shutil.rmtree() class. It doesn't actually ignore
05928              any errors. It just forces the function to delete
05929              the files and or folders and more importantly, to
05930              not print out any errors. It's a boolean variable
05931              and we set it to true to ensure that it does in fact
05932              force the deletion of the folder.
05933         Members
05934         -------
05935             sys.platform.startswith('string')
05936                 We use this function to detect the users operating
05937                 system. It will determine whether or not you're using
05938                 a windows or linux based operating system. It doesn't
05939                 search for a specific version. It just ensures that
05940                 you're using one of them. This allows us to make this
05941                 program cross platform.
05942             shutil.rmtree(path, optional_boolean)
05943                We use this function to preform our deletion operations.
05944                By default it makes use of the native systems api to
05945                preform this operation. We need to suuply it a path,
05946                this where our temp variable comes in. You'll remember
05947                that we stored the path to the directory in it. We
05948                also suply this function with a boolean variable
05949                ignore_errors and set it to True. As mentioned above
05950                this simply ensures that the folder gets deleted,
05951                whether it's empty (this is what it checks for) or not
05952                and prevents it from spitting out any warnings or errors
05953                from the system. This bool vairable is by default set
05954                to False.
05955             self.ids.view_port.text
05956                 This is the view_port TextInput widget in the
05957                 kv code. We use this widget to display certain
05958                 warnings and conversations to the user. In this
05959                 case when the folder has been deleted we inform the
05960                 user by printing out the folders path and stating
05961                 that it has been deleted.
05962             Private members
05963             ---------------
05964                 self.__append_file(string, filepath)
05965                    This function is a member of SentienceScreen(). We
05966                    call this function to append error messages if they
05967                    occur to an error log. We supply the function,
05968                    the exception, what occured, and the date and time
05969                    that it occured to this fileself.
05970             Returns
05971             -------
05972                 None
05973             Exceptions
05974             ----------
05975                 IOError
05976                     The IOError can occur due to many reasons.
05977                     My primary concern is file manipulation. The
```

```
05978                   improper opening/closing/writing to files. If
05979                   the exception occurs it's handled and logged; in
05980                   an error log text file.
05981               FileNotFoundError
05982                   This can occur in a variety of ways however my
05983                   primary concern is that file path the user selected
05984                   is broken. Resulting in an File Not Found error.
05985                   If this occurs it's handled and logged to an error
05986                    file text log.
05987           Notes
05988           -----
05989               First we check to make sure the user is running a
05990               viable operating system. Rather, one that's
05991               compatible with this program. We then execute the
05992               appropriate code.
05993
05994               We create a variable named temp and sore the absolute
05995               path of the directory (folder) that we're going to
05996               delete.
05997
05998               We then call shutil.rmtree() to make access of the
05999               systems native api to delete the directory. Once deleted
06000               we display a message which includes the full directory
06001               path and a string stating that the folder has been
06002               deleted.
06003
06004               If any errors occur we record and log them to an error
06005               log.
06006       '''
06007       try:
06008           if sys.platform.startswith('linux'):
06009               temp = '/home/' + str(os.getlogin()) + '/.SentienceFiles/'
06010               shutil.rmtree(temp, ignore_errors=True)
06011               self.ids.view_port.text = temp + ' and all of its contents have been deleted'
06012               App.get_running_app().stop()
06013           elif sys.platform.startswith('win'):
06014               temp = 'C://SentienceFiles//'
06015               shutil.rmtree(temp, ignore_errors=True)
06016               self.ids.view_port.text = temp + ' and all of its contents have been deleted'
06017               App.get_running_app().stop()
06018       except IOError as a:
06019           if sys.platform.startswith('linux'):
06020               self.__append_file('\n' + 'Function: delet_all ' + '\nIOError: ' + str(a) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(os.getlogin()) +
       '/.SentienceFiles/Error Logs.txt')
06021           elif sys.platform.startswith('win'):
06022               self.__append_file('\n' + 'Function: delete_all ' + '\nIOError: ' + str(a) + '
      \nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), 'C://SentienceFiles//Error
       Logs.txt')
06023           return None
06024       except FileNotFoundError as b:
06025           if sys.platform.startswith('linux'):
06026               self.__append_file('\n' + 'Function: delete_all ' + '\nFileNotFoundError: ' +
      str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '/home/' + str(
      os.getlogin()) + '/.SentienceFiles/Error Logs.txt')
06027           elif sys.platform.startswith('win'):
06028               self.__append_file('\n' + 'Function: delete_all ' + '\nFileNotFoundError: ' +
      str(b) + '\nDate - Time:' + str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')), '
      C://SentienceFiles//Error Logs.txt')
06029           return None
06030
06031
06032
06033   def __set_thinking_text(self, bool):
06034       '''
06035       __Set_thinking_text(self, bool)
06036
06037       Parameters
06038       ----------
06039           param1 : self
06040               Denotes this as being a member of the SentienceScreen()
06041               class.
06042
06043           param2 : bool
06044               The boolean variable contains the dertmining factor
06045               for how the text will be set and what color that text
06046               will be in the self.notification_widget. It recieves
06047               this information from the
06048               self.__currently_thinking() function.
06049       Attributes
06050       ----------
06051           bool
06052               if bool == True then the chatbot is about to begin
06053               generating a response for the user. We then set the
06054               text field of self.notification_widget to
06055               '...Thinking...' and set the foreground to red. If
06056               it's False we know then that the chatbot just
```

```
06057                finished generating a response. We then set the text
06058                field of self.notification_widget to '...Inactive...'
06059                with a blue foreground.
06060        Members
06061        -------
06062        self.notification_widget.text
06063            We use this to set the text field of the
06064            self.notification_widget TextInput widget to the
06065            appropriate text based on the preceeding conditions. This
06066            is a member of the SentienceScreen() class.
06067
06068        self.ids.notification_widget.foreground_color
06069            We use this to give the text a color. The color is relative
06070            to the programs current status. If the program is
06071            '...Thinking...' then the text will be red. If the program
06072            is '...Inactive...' then the text will be blue. This
06073            function is a member of the SentienceScreen() class.
06074
06075        kivy.utils.get_color_from_hex('hex string')
06076            This is a member of the Kivy base class. We use this
06077            function to convert a hexadecimal string to a compatiable
06078            color read as an integer by the self.ids.notification_widget
06079            TextInput widget.
06080        Private Members
06081        --------------
06082            None
06083        Exceptions
06084        ----------
06085            None
06086        Returns
06087        -------
06088            None
06089        Notes
06090        -----
06091            We call this function to change the self.notification_widget
06092            TextInput widgets text field to either '...Thinking...' or
06093            '...Inactive...'. This status is determined when the chatbot
06094            is generating or finished generating a response. If the
06095            chatbot is generating a response it's thinking. If it has
06096            finished generating a reponse then it's set to inactive.
06097
06098            We then change the color of the text using a hex string
06099            which is then read as an integer byt the
06100            self.notification_widget.foreground_color property.
06101            If the program is thinking then the text is made red.
06102
06103            If the program is inactive then the text is made blue.
06104        '''
06105        if bool == True:
06106            self.ids.notification_widget.text = '...Thinking...'
06107            self.ids.notification_widget.foreground_color = kivy.utils.get_color_from_hex('FF0000')
06108        elif bool == False:
06109            self.ids.notification_widget.text = '...Inactive...'
06110            self.ids.notification_widget.foreground_color = kivy.utils.get_color_from_hex('00FFFF')
06111
06112
06113
06114    def __currently_thinking(self, bool):
06115        '''
06116        __currently_thinking(self, bool)
06117
06118        param1: self
06119            Denotes this as being a member of the SentienceScreen()
06120            class.
06121
06122        param2 : bool
06123            A boolean variable is passed to this function and
06124            then used to dertime the text in the self.notification_widget
06125            TextInput Widget.
06126
06127        Attributes
06128        ----------
06129            bool
06130                bool is used to store the boolean variable
06131                passed to this function upon its function call.
06132        Members
06133        -------
06134            None
06135        Privat Members
06136        --------------
06137            self.__is_thinking
06138                self.__is_thinking is a member of the SentienceScreen()
06139                class. We use this to monitor the status the 'thinking'
06140                status of the program and to switch it on and off. If
06141                it's on self.notification_widget has its text field set
06142                to '...Thinking...' with a red foreground. If it's off
06143                self.notification_widget has its text field set to
```

```
06144                 '...Inactive...' with a blue foregrounf.
06145             self.__set_thinking_text(bool)
06146                 Once we know if the program is currently thinking
06147                 we call this function which is a member of the
06148                 SentienceScreen() class. To actually change
06149                 the text field of the self.notification_widget.
06150         Notes
06151         -----
06152             We call this function to check to see if the chatbot
06153             is about to begin generating a response to the user.
06154
06155             If the chatbot is about to begin it's generation process
06156             we set the self.__is_thinking variable to True, we then
06157             call self.__set_thinking_text(bool) to change the
06158             text field of self.notification_widget to '...Thinking...'
06159             with a red foreground.
06160
06161             If the chatbot is done generating a resposne we set
06162             the variable self.__is_thinking to False, and then call
06163             self.__set_thinking_text(bool) to change the text field
06164             of the self.notification_widget to '...Inactive...' with
06165             a blue foreground color.
06166         '''
06167         if bool == True:
06168             self.__is_thinking = True
06169             return self.__set_thinking_text(True)
06170         elif bool == False:
06171             self.__is_thinking == False
06172             return self.__set_thinking_text(False)
06173
06174
06175
06176 class ActionInput(TextInput, ActionItem):
06177     '''
06178     Creates a class for the ActionInput(TextInput, ActionItem):
06179
06180     This TextInput will be placed on the ActionBar()
06181
06182     '''
06183     pass
06184
06185
06186 class sentienceScreenManager(ScreenManager):
06187     '''
06188     sentienceScreenManager(ScreenManager)
06189     container/manager for SentienceScreen:
06190     '''
06191     pass
06192
06193
06194 root_widget = Builder.load_string('''
06195 #:import Config kivy.config
06196 #:import Window kivy.core.window
06197 #:import Clock kivy.clock
06198 #:import ActionBar kivy.uix.actionbar
06199 #:import Animation kivy.animation.Animation
06200 #:import hex kivy.utils.get_color_from_hex
06201
06202 sentienceScreenManager:
06203     SentienceScreen:
06204
06205 <ToolTipLabel@Label>:
06206     size_hint: None, None
06207     text_size: self.width, None
06208     height: self.texture_size[1]
06209
06210     canvas.before:
06211         Color:
06212             rgb: 0,0,0
06213         Rectangle:
06214             size: self.size
06215             pos: self.pos
06216
06217 <SentienceScreen>:
06218     name: 'main_screen'
06219
06220     TextInput:
06221         id: view_port
06222         size_hint: .5, .85
06223         pos_hint: {'x': .5, 'y': .0 }
06224         multiline: True
06225         hint_text: 'Conversation log'
06226         readonly: True
06227     TextInput:
06228         id: user_input
06229         size_hint: .5, .05
06230         pos_hint: {'x': 0, 'y': .80 }
```

```
06231         multiline: False
06232         hint_text: 'Type your question/response'
06233         on_text_validate: root.start_get_response_thread()
06234     TextInput:
06235         id: notification_widget
06236         size_hint: .3, .05
06237         pos_hint: {'x': .80, 'y': .85}
06238         multiline: False
06239         readonly: True
06240         text: '...Inactive...'
06241         foreground_color: hex('00FFFF')
06242         background_color: (0,0,0,0)
06243     TextInput:
06244         text: "Please allow 20-90 seconds for a response. Don't interact with the program until its status
       is ...Inactive... in blue text."
06245         size_hint: 1, .05
06246         pos_hint: {'x': 0, 'y': .85}
06247         font_size: '12sp'
06248         readonly: True
06249         foreground_color: (1,1,0,1)
06250         background_color: (0,0,0,0)
06251
06252     ActionBar:
06253         id: menu_bar
06254         pos_hint: {'top':1}
06255         ActionView:
06256             use_separator: True
06257             ActionPrevious:
06258                 title: 'Menu'
06259                 with_previous: False
06260             ActionOverflow:
06261             ActionSeparator:
06262                 separtor_width: 10
06263                 opacity: 0
06264             ActionButton:
06265                 id: display_convo
06266                 text: 'Display Conversations'
06267                 on_press: root.display_user_conversation()
06268             ActionButton:
06269                 id: open_settings
06270                 text: 'Settings'
06271                 on_press: app.open_settings()
06272             ActionSeparator:
06273                 separtor_width: 10
06274                 opacity: 0
06275             ActionButton:
06276                 id: record_user
06277                 text: 'Record'
06278                 on_release: root.start_voice_response_thread()
06279             ActionSeparator:
06280                 separtor_width: 10
06281                 opacity: 0
06282
06283 <PrintDialog>:
06284     BoxLayout:
06285         size: root.size
06286         pos: root.pos
06287         orientation: 'vertical'
06288         FileChooserListView:
06289             id: filechooser
06290         BoxLayout:
06291             size_hint_y: None
06292             height: 30
06293             Button:
06294                 text: 'Cancel'
06295                 on_release: root.Cancel()
06296             Button:
06297                 text: 'Print File'
06298                 on_release: root.print_files(filechooser.path, filechooser.selection)
06299
06300 <DeleteDialog>:
06301     BoxLayout:
06302         size: root.size
06303         pos: root.pos
06304         orientation: 'vertical'
06305         FileChooserListView:
06306             id: deletion_chooser
06307         BoxLayout:
06308             size_hint_y: None
06309             height: 30
06310             Button:
06311                 text: 'Cancel'
06312                 on_release: root.Cancel()
06313             Button:
06314                 text: 'Delete File'
06315                 on_release: root.delete_file(deletion_chooser.path, deletion_chooser.selection)
06316 ''')
```

```
06317
06318
06319 class SentienceApp(App):
06320
06321     def build(self):
06322         '''
06323             Kivy App() class is the core class that creates the
06324              main window and runs the program.
06325
06326              You'll also see that I've created a series of variables
06327              which are used to store the data contained in
06328              Sentience.ini
06329
06330              The data stored in these variables is loaded into them
06331              everytime that this program is run so that they always
06332              reflect the accurate settings.
06333         '''
06334         self.title = 'Honors Project: Sentience'
06335         self.settings_cls = SettingsWithSidebar #
06336         self.config.items('settings_menu')
06337         self.sentience = root_widget.get_screen('main_screen')
06338         self.use_kivy_settings = False
06339         self.sentience.caprica_speak('Loading your settings now.')
06340         self.is_audio_on = self.config.get('settings_menu', 'enable_audio')
06341         self.increased_rate_of_speech = self.config.get('settings_menu', '
    increase_rate_of_speech')
06342         self.decreased_rate_of_speech = self.config.get('settings_menu', '
    decrease_rate_of_speech')
06343         self.is_voice_on = self.config.get('settings_menu', 'enable_voice')
06344         self.print_status = self.config.get('settings_menu', 'get_file_printed')
06345         self.delete_file_status = self.config.get('settings_menu', 'get_file_deleted')
06346         self.delete_all_status = self.config.get('settings_menu', 'delete_everything')
06347         self.user_exists = self.config.get('settings_menu', 'create_user')
06348         self.gender_exists = self.config.get('settings_menu', 'create_gender')
06349         self.age_exists = self.config.get('settings_menu', 'create_age')
06350         self.clear_screen_status = self.config.get('settings_menu', 'clear_screen')
06351         self.file_creation_status = self.config.get('settings_menu', 'write_user_data')
06352         self.load_settings()
06353         return root_widget
06354
06355
06356
06357     def load_settings(self):
06358         '''
06359             This is fairly straightforward. When the program launches
06360             we look at the data contained in Sentience.ini which holds
06361             the values from our Ssettings panel. Everytime that the
06362             user changes a setting it's written to Sentience.ini
06363
06364             Based on those settings we set or ignore specific variables
06365             from SentienceScreen(). We use this to maintain continuity
06366             between the end of the program and it being restarted.
06367         '''
06368         if '1' in self.is_audio_on:
06369             self.sentience.audio_enabled = True
06370             self.sentience.audio_disabled = False
06371             self.sentience.set_enable_disable_audio()
06372         elif '0' in self.is_audio_on:
06373             self.sentience.audio_enabled = False
06374             self.sentience.audio_disabled = True
06375             self.sentience.set_enable_disable_audio()
06376         if int(self.increased_rate_of_speech) > 0:
06377             self.sentience.increase_rate_of_speech(int(self.
    increased_rate_of_speech))
06378         elif int(self.increased_rate_of_speech) <=0:
06379             pass
06380         if int(self.decreased_rate_of_speech) > 0:
06381             self.sentience.decrease_rate_of_speech(int(self.
    decreased_rate_of_speech))
06382         elif int(self.decreased_rate_of_speech) <=0:
06383             pass
06384         if '1' in self.is_voice_on:
06385             self.sentience.voice_enabled = True
06386             self.sentience.voice_disabled = False
06387             self.sentience.set_enable_disable_voice()
06388         elif '0' in self.is_voice_on:
06389             self.sentience.voice_enabled = False
06390             self.sentience.voice_disabled = True
06391             self.sentience.set_enable_disable_voice()
06392         if 'print file' in self.print_status:
06393             self.config.set('settings_menu', 'get_file_printed', 'None')
06394             self.write()
06395         elif 'None' in self.print_status:
06396             pass
06397         if 'delete file' in self.delete_file_status:
06398             self.config.set('settings_menu', 'get_file_deleted', 'None')
06399             self.config.write()
```

```
06400         elif 'delete file' or 'None' not in self.delete_file_status:
06401             self.config.set('settings_menu', 'get_file_deleted', 'None')
06402             self.config.write()
06403         if 'delete all' in self.delete_all_status:
06404             self.config.set('settings_menu', 'delete_everything', 'None')
06405             self.config.write()
06406         elif 'delete all' or 'None' not in self.delete_all_status:
06407             self.config.set('settings_menu', 'delete_everything', 'None')
06408             self.config.write()
06409         if 'None' not in self.user_exists:
06410             self.set_username(self.user_exists)
06411         elif 'None' in self.user_exists:
06412             pass
06413         if 'None' not in self.gender_exists:
06414             self.set_gender(self.gender_exists)
06415         elif 'None' in self.gender_exists:
06416             pass
06417         if 'None' not in self.age_exists:
06418             self.set_age(self.age_exists)
06419         elif 'None' in self.age_exists:
06420             pass
06421         if 'yes' or 'Yes' in self.clear_screen_status:
06422             self.config.set('settings_menu', 'clear_screen', 'None')
06423             self.config.write()
06424         elif 'yes' or 'Yes' or 'None' not in self.clear_screen_status:
06425             self.config.set('settings_menu', 'clear_screen', 'None')
06426             self.config.write()
06427         if 'write file' in self.file_creation_status:
06428             self.config.set('settings_menu', 'write_user_data', 'None')
06429             self.config.write()
06430         elif 'write file' or 'None' not in self.file_creation_status:
06431             self.config.set('settings_menu', 'write_user_data', 'None')
06432             self.config.write()
06433
06434
06435
06436     def build_config(self, config):
06437         '''
06438             We call this to build the menu. We're establishing
06439             default values which we'll then load in from our json
06440             string. This is what creats the actuall widgets and
06441             links them to the keys, provides default values, and
06442             registers them.
06443         '''
06444         config.setdefaults('settings_menu', {
06445                             'enable_audio': False,
06446                             'increase_rate_of_speech': 0,
06447                             'decrease_rate_of_speech': 0,
06448                             'enable_voice': False,
06449                             'get_file_printed': None,
06450                             'select_your_os': None,
06451                             'get_file_deleted': None,
06452                             'delete_everything': None,
06453                             'create_user': None,
06454                             'create_gender': None,
06455                             'create_age': None,
06456                             'clear_screen': None,
06457                             'write_user_data': None
06458                             })
06459
06460
06461
06462     def build_settings(self, settings):
06463         '''
06464             Here we add the settings panel as widget in the form
06465             of a json object. We pass it the name of our panel,
06466             Sentience Settings, our self.config from build_config
06467             and the json strings which contains all of the data in
06468             self.config. This setups, create and adds the settings
06469             panel to the screen.
06470         '''
06471         settings.add_json_panel('Sentience Settings', self.config, data = my_settings)
06472
06473
06474
06475     def on_start(self):
06476         '''
06477             This function creates a cProfiler() to help us
06478             diagnose potential issues.
06479         '''
06480         self.profiler = cProfile.Profile()
06481         self.profiler.enable()
06482
06483
06484
06485     def on_stop(self):
06486         '''
```

```
06487             When the program is exited the profiler is stopped
06488             and the .profile file containing the data that's
06489             been accumulated to help test the program is output
06490             to a file named SentiencePRofile.profile
06491         '''
06492         self.profiler.disable()
06493         self.profiler.dump_stats('SentienceProfile.profile')
06494 
06495 
06496 
06497     def warning_removal(self, dt):
06498         '''
06499             A simple function to clear the contents of
06500             self.sentience.ids.view_port Widget.
06501         '''
06502         self.sentience.ids.view_port.text = ''
06503 
06504 
06505 
06506     def set_username(self, value):
06507         '''
06508             This function is called to set the first key of
06509             self.sentience.user_profile[1] dictionary to
06510             value. value is then stored in self.sentience.username
06511             self.sentience.master_log string is then cleared to
06512             ensure a new user experience is created. We then create
06513             the user profile which basically just reads the users
06514             input username which is stored in value.
06515         '''
06516         self.sentience.user_profile[1] = value
06517         self.sentience.username = value
06518         self.sentience.master_log = ''
06519         self.sentience.create_user_profile()
06520 
06521 
06522 
06523     def set_gender(self, value):
06524         '''
06525             We call this function to set the gender supplied to
06526             value when the users modifies the Gender setting in
06527             the settings menu. We store the gender in value in
06528             self.sentience.user_priofile[3]. We then clear the
06529             self.sentience.master_log string to ensure a new
06530             experience has been created for the current user.
06531         '''
06532         self.sentience.user_profile[3] = value
06533         self.sentience.master_log = ''
06534 
06535 
06536 
06537     def set_age(self, value):
06538         '''
06539             We call this function to set the age supplied to
06540             value when the users modifies the age setting in
06541             the settings menu. We store the gender in value in
06542             self.sentience.user_priofile[2]. We then clear the
06543             self.sentience.master_log string to ensure a new
06544             experience has been created for the current user.
06545         '''
06546         self.sentience.user_profile[2] = value
06547         self.sentience.master_log = ''
06548 
06549 
06550 
06551     def on_config_change(self, config, section, key, value):
06552         '''
06553             Simple vent monitor. Every time that the user changes
06554             a setting on the Settings panel, this function is called.
06555 
06556             It recevies the section (menu), the key (specific option),
06557             and the value which is what the current option has been
06558             changed to. From there we change the variables in the
06559             SentienceScreen() class.
06560         '''
06561         if section == 'settings_menu':
06562             if section == 'settings_menu' and key == 'enable_audio' and value == '1':
06563                 self.sentience.audio_enabled = True
06564                 self.sentience.audio_disabled = False
06565                 self.sentience.set_enable_disable_audio()
06566                 self.config.set('settings_menu', 'enable_audio', value)
06567                 self.config.write()
06568             elif section == 'settings_menu' and key == 'enable_audio' and value == '0':
06569                 self.sentience.audio_enabled = False
06570                 self.sentience.audio_disabled = True
06571                 self.sentience.set_enable_disable_audio()
06572                 self.config.set('settings_menu', 'enable_audio', value)
06573                 self.config.write()
```

```
06574             elif section == 'settings_menu' and key == 'enable_voice' and value == '1':
06575                 self.sentience.voice_enabled = True
06576                 self.sentience.voice_disabled = False
06577                 self.sentience.set_enable_disable_voice()
06578                 self.config.set('settings_menu', 'enable_voice', value)
06579                 self.config.write()
06580             elif section == 'settings_menu' and key == 'enable_voice' and value == '0':
06581                  self.sentience.voice_enabled = False
06582                  self.sentience.voice_disabled = True
06583                  self.sentience.set_enable_disable_voice()
06584                  self.config.set('settings_menu', 'enable_voice', value)
06585                  self.config.write()
06586             elif section == 'settings_menu' and key == 'create_user' and 'None' not in value:
06587                 self.set_username(value)
06588                 self.config.set('settings_menu', 'create_user', value)
06589                 self.config.write()
06590             elif section == 'settings_menu' and key == 'create_gender' and 'None' not in value:
06591                 self.set_gender(value)
06592                 self.config.set('settings_menu', 'create_gender', value)
06593                 self.config.write()
06594             elif section == 'settings_menu' and key == 'create_age' and 'None' not in value:
06595                 self.set_age(value)
06596                 self.config.set('settings_menu', 'create_age', value)
06597                 self.config.write()
06598             elif section == 'settings_menu' and key == 'clear_screen' and 'yes' or 'Yes' in value:
06599                 self.sentience.clear_viewport()
06600                 self.config.set('settings_menu', 'clear_screen', 'None')
06601                 self.config.write()
06602             elif section == 'settings_menu' and key == 'get_file_deleted' and 'delete file' in value:
06603                 self.sentience.open_delete_file_dialog()
06604                 self.config.set('settings_menu', 'get_file_deleted', 'None')
06605                 self.config.write()
06606             elif section == 'settings_menu' and key == 'delete_everything' and 'delete all' in value:
06607                 self.config.set('settings_menu', 'delete_everything', 'None')
06608                 self.config.write()
06609                 self.sentience.delete_all()
06610             elif section == 'settings_menu' and key == 'get_file_printed' and 'print file' in value:
06611                 self.sentience.open_print_file_dialog()
06612                 self.config.set('settings_menu', 'get_file_printed', 'None')
06613                 self.config.write()
06614             elif section == 'settings_menu' and key == 'write_user_data' and 'write file' in value:
06615                 self.sentience.write_logs()
06616                 self.config.set('settings_menu', 'write_user_data', 'None')
06617                 self.config.write()
06618             elif section == 'settings_menu' and key == 'increase_rate_of_speech':
06619                 if int(value) >= 1:
06620                     self.sentience.increase_rate_of_speech(int(value))
06621                     self.config.set('settings_menu', 'increase_rate_of_speech', value)
06622                     self.config.write()
06623                 elif int(value) <=0:
06624                     self.sentience.increase_rate_of_speech(20)
06625                     self.config.set('settings_menu', 'increase_rate_of_speech', '20')
06626                     self.config.write()
06627             elif section == 'settings_menu' and key == 'decrease_rate_of_speech':
06628                 if int(value) >= 1:
06629                     self.sentience.decrease_rate_of_speech(int(value))
06630                     self.config.set('settings_menu', 'decrease_rate_of_speech', value)
06631                     self.config.write()
06632                 elif int(value) <=0:
06633                     self.sentience.decrease_rate_of_speech(20)
06634                     self.config.set('settings_menu', 'decrease_rate_of_speech', '20')
06635                     self.config.write()
06636
06637
06638
06639 if __name__ == '__main__':
06640     ''' Main Loop
06641     '''
06642     SentienceApp().run()
```

7.3 SettingsMenu.py File Reference

Namespaces

- SettingsMenu

Variables

- SettingsMenu.my_settings

7.4 SettingsMenu.py

```
00001 import json
00002
00003 my_settings = json.dumps([
00004     {
00005         "type": "bool",
00006         "title": "Enable Audio",
00007         "desc": "Turns on the chatbots ability to verbally speak it's response to the user. To turn it on
      click the button labeled on. To turn it off click the button labeled off.",
00008         "section": "settings_menu",
00009         "key": "enable_audio"
00010     },
00011     {
00012         "type": "numeric",
00013         "title": "Increase rate of speech",
00014         "desc": "Increase the chatbots rate of speech by the number that you supply. If you set this to 40,
      you've increased the rate by 40 words per minute spoken. Which means if you want to lower the rate by 40,
      you need to input 40 in the 'Decrease rate of speech' box. ",
00015         "section": "settings_menu",
00016         "key": "increase_rate_of_speech"
00017     },
00018      {
00019         "type": "numeric",
00020         "title": "Decrease rate of speech",
00021         "desc": "Decrease the chatbots rate of speech by the number that you supply. If you set this to 40,
      you've decreased the rate by 40 words per minute spoken. Which means if you want to increase the rate by
      40, you need to input 40 in the 'Increase rate of speech' box. ",
00022         "section": "settings_menu",
00023         "key": "decrease_rate_of_speech"
00024     },
00025     {
00026         "type": "bool",
00027         "title": "Enable Voice",
00028         "desc": "Allows the user the ability to use their microphone to speak with the chatbot. To turn it
      on click the button labeled on. To turn it off click the button labeled off",
00029         "section": "settings_menu",
00030         "key": "enable_voice"
00031     },
00032     {
00033         "type": "string",
00034         "title": "Print File",
00035         "desc": "To Locate and select a file to print, simply type 'print file' into this box and then
      click the okay button. This will open a file browser for you to navigate to and select a file to print. Please
      note that the file will not print untill after the program has closed.",
00036         "section": "settings_menu",
00037         "key": "get_file_printed"
00038     },
00039     {
00040         "type": "string",
00041         "title": "Delete File",
00042         "desc": "To locate and select a file to delete simply type 'delete file' into the text box and then
      click okay. This will open a file browser which will allow you to navigate to and select a file for
      deletion.",
00043         "section": "settings_menu",
00044         "key": "get_file_deleted"
00045     },
00046     {
00047         "type": "string",
00048         "title": "Delete All Files",
00049         "desc": "To delete all files and folders generated by this program, simply type the text 'delete
      all' into the box and then click okay. Note: Deleting all files and folders generated by this program will
      cause the program to exit. To re-generate these files just run the program again.",
00050         "section": "settings_menu",
00051         "key": "delete_everything"
00052     },
00053     {
00054         "type": "string",
00055         "title": "Enter username",
00056         "desc": "Create a username if you want to, by entering it into this box and clicking the okay
      button when prompted. You're free to enter what ever you want even if it's a joke. However, being truthful may
      give more meaning to your conversation.",
00057         "section": "settings_menu",
00058         "key": "create_user"
00059     },
00060     {
00061         "type": "string",
00062         "title": "Enter Gender",
00063         "desc": "Enter your gender if you want to, by typing it into this box and clicking the okay button
      when prompted. You're free to enter what ever you want even if it's a joke. However, being truthful may give
      more meaning to your conversation.",
00064         "section": "settings_menu",
00065         "key": "create_gender"
00066     },
00067     {
00068         "type": "string",
```

```
00069         "title": "Enter your age",
00070         "desc": "Enter your age if you want to, by typing it into this box and clicking the okay button
      when prompted. You're free to enter what ever you want even if it's a joke. However, being truthful may give
      more meaning to your conversation.",
00071         "section": "settings_menu",
00072         "key": "create_age"
00073     },
00074     {
00075         "type": "string",
00076         "title": "Clear Conversation Logs",
00077         "desc": "If you wish to clear the text in the large box labeled 'Conversation log', simply type the
      word 'yes' into this box and then click the okay button. To do this again simply click on this section,
      delete the text click okay, then re-type the word 'yes'. This is a security precausion",
00078         "section": "settings_menu",
00079         "key": "clear_screen"
00080     },
00081     {
00082         "type": "string",
00083         "title": "Write User Conversation",
00084         "desc": "If you'd like to write a text file that contains your entire conversation with the
      chatbot. Just type 'write file' into this text box and then click the okay button. The file will be named either
      User_Conversations.txt or, if you gave a username, Username_Conversation.txt. Where Username is the name that
      you supplied.",
00085         "section": "settings_menu",
00086         "key": "write_user_data"
00087     },
00088 ])
```

Index