PIPSTA107 –SCRATCH LISTENER PYTHON CODE TUTORIAL

Contents

Revision History

Revision	Author	Date	Description
1.0	AH	26/11/14	First Release

Difficulty Level:

- This tutorial focuses mainly on communication between Scratch and Python, and so complexity is limited.

Time to Complete:

- Some aspects are complex and require significant time to explain and comprehend.
- The range of functionality offered in the Scratch Listener makes this tutorial fairly lengthy.

Who Should Read This Document

It is assumed that the reader has worked though some of the earlier tutorials, and has some understanding of (or is at least happy to 'ring-fence' their understanding of) aspects such as USB Enumeration, graphic manipulation by Pillow Imaging Library and the conversion from a bitmap image to a format suitable for transmission to the printer. However, the range of printing options presented in this script mean that it is possible to 'tinker' with the listener to add functionality cribbed from pre-existing code, to –for example—issue a different graphic type.

The focus of this tutorial will be rather less on printing aspects and more on the functionality of the listener and its underlying *state machine*.

It is therefore hoped that this tutorial will be useful to developers and brave experimenters alike!

Prerequisites

This guide is intended for users who have completed the mechanical build of their Pipsta and have successfully performed the first-time setup concluding in a simple print-out, as per:

- ***PIPSTA002 - Pipsta B+ Assembly Instructions*** or
- ***PIPSTA003 - Pipsta A+ Assembly Instructions***
  and
- ***PIPSTA004 – Pipsta First-Time Setup***

It is expected that you have the following items:

- Your assembled, working Pipsta
- Power supply for Raspberry Pi (5v, ideally no less than 2.0A rated)
- USB A to Micro B cable assembly (if not integrated into Raspberry Pi power supply)
- Micro SD Card configured as per the *Pipsta First-Time Setup tutorial*
- Consider an additional, reserve Micro SD Card!
- Access to two mains sockets for powering the Raspberry Pi and Ap1400 printer
- USB Keyboard
- USB Mouse
- A Wi-Fi connection as per the *Pipsta First-Time Setup tutorial*
- Video/Monitor lead:
  - HDMI lead *or*
  - 3.5mm 4 pole jack plug to RCA composite cable *or*
  - HDMI to VGA adaptor and VGA cable
- Computer monitor or television (with HDMI, component video or VGA input as above)
- You may also want to confirm the 1D barcodes produced in this example are scannable by downloading a smartphone app. This is not a requirement as the human-readable code appears underneath each barcode.
- If you have not already done so, install ***scratchpy*** as follows:

Additional Configuration Step

- Open LXTerminal
- At the $ prompt, enter:
  sudo pip install scratchpy
  to install the module that permits communications between Scratch and Python. This is a one-time operation, and will not need to be repeated once installed.

How it Works

The library *scratchpy* is at the heart of the Pipsta Python Listener. Whilst Scratch can use *broadcasts* to signal between event-driven functions within Scratch, *scratchpy* makes it possible for a *Python script* to listen for these broadcasts, and respond accordingly.

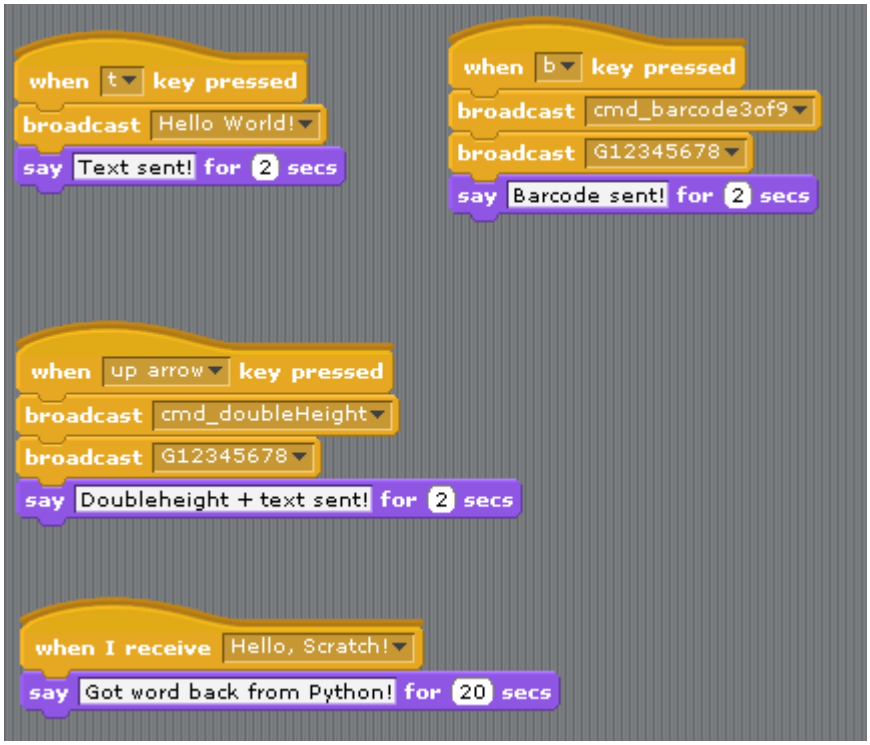
In this implementation, Scratch is used to send a sequence of broadcasts at the end of an educational game, in order to print a certificate. To make this certificate as configurable as possible *within Scratch* and thereby allow considerable modification by users with no Python knowledge, a host of specific broadcasts are catered-to by the Python listener.

In order to create a firm foundation upon which to build, the Pipsta Scratch code is presented in three stages of development:

- simplelisten (for use with *simple_scratch.sb*)
- improvedlisten (for use with *improved_scratch.sb*)
- certificate (for use with *NumberBonds.sb)*

simplelisten.py and simple_scratch.sb

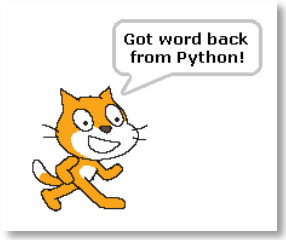
1) In Scratch (V1.4), load ***simple_scratch.sb***
2) The Scripts window should now show the following:



3) In LXTerminal, type the following at the $ prompt:

   python simplelisten.py

4) Back in Scratch, you should see that the cat now has a speech bubble:



5) This demonstrates communications from the Python script to Scratch using *broadcasts*. The Python code responsible is:

```
scratch_conn = scratch.Scratch()
scratch_conn.broadcast("Hello, Scratch!")
```

6) The first line establishes the connection to Scratch, the second exercises this connection with a broadcast.
7) Just as Python can broadcast to Scratch, Scratch can broadcast to Python. Back in Scratch, click on the following block or press 'b':

8) In the LXTerminal window, you should see:

```
barcode setup
G12345678
```

…though the content of this message is arbitrary and could in effect be anything. Note that –for the purposes of this simple introduction—the Pipsta printer is not yet being asked to print anything: the handlers are just *stubs* which show how the handling logic works. The Pipsta printer functionality will start to be introduced in the next script.

9) The code responsible for displaying this message is in the following block:

```
try:
    for msg in listen(scratch_conn):
        if msg[0] == 'broadcast':
            # Handle able command types
            if msg[1] == 'cmd_barcode3of9':
                print('barcode setup')
            elif msg[1] == 'cmd_doubleHeight':
                print('double height setup')
            else:
                print(msg[1])
except KeyboardInterrupt:
    # Normal way of exiting app, anything else will raise an error
    pass
finally:
    # If the failure is expected or not - close down gracefully
    scratch_conn.disconnect()
```

10) The *for loop* loops forever, continually calling **listen(scratch)**. Broadcast messages consist of a two element array:

msg[0] = “broadcast”
msg[1] = message

The above code checks that **msg[0]** is indeed a broadcast before then looking for specific messages in **msg[1]** and handling them appropriately.

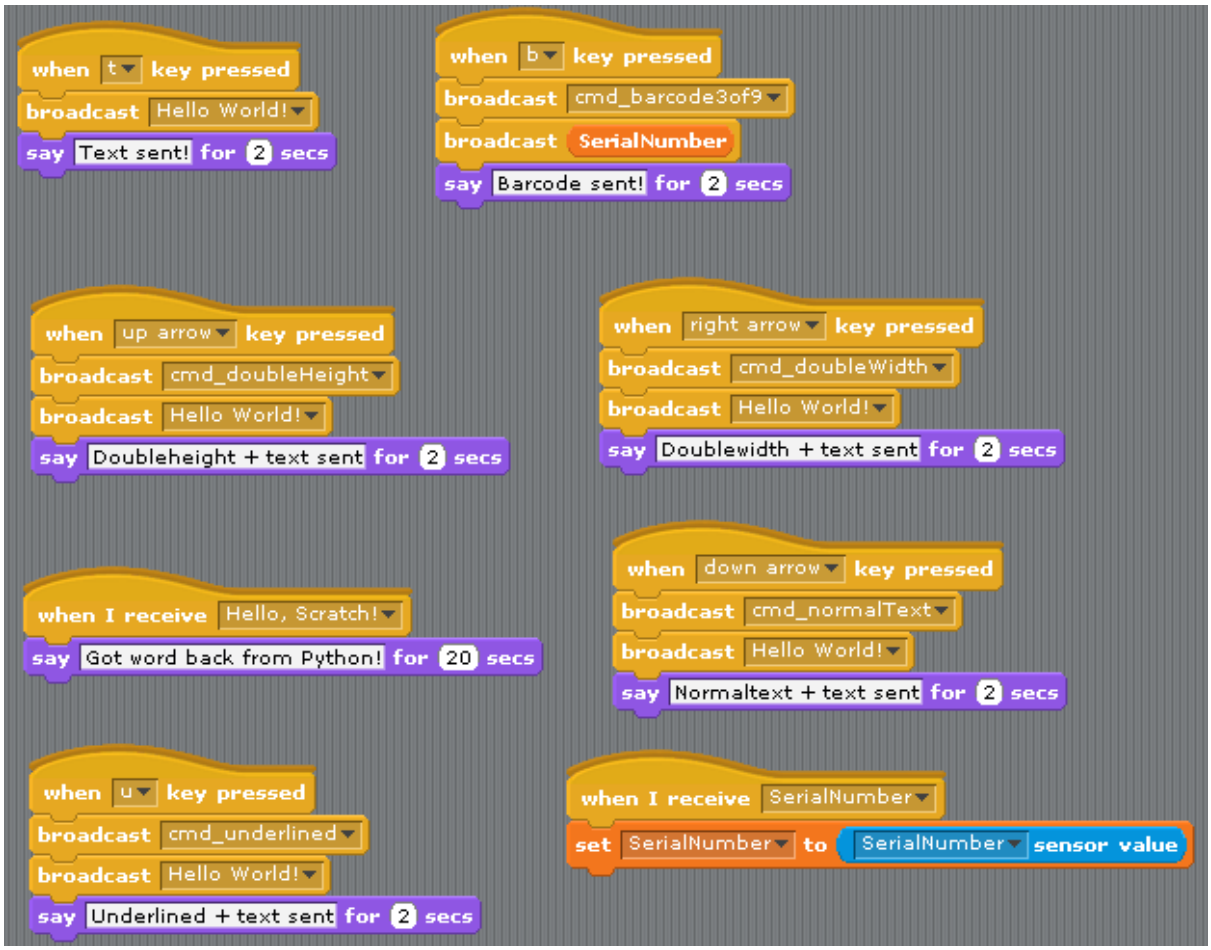
11) Try pressing [up arrow] and [T] to see their handling.
12) It can be seen that adding handlers for other broadcasts is simply a matter of adding another *if statement* and corresponding handling.
13) Exit Scratch, noting that the Python script will also terminate at this time.

Improvedlisten.py and improved_scratch.sb

In the previous example Scratch program and Python Listener, we demonstrated Python’s ability to respond to broadcasts from Scratch. The handlers in primitive version were *stubbed* i.e. did not contain any code for communicating with the Pipsta printer.

In improvedlisten.py, printing functionality will be introduced. Once again, this implementation has been kept as simple as possible for clarity.

1) In Scratch (V1.4), load ***improved_scratch.sb***
2) The Scripts window should now show the following:



3) You should notice that there are a few more additional functions now as compared with the previous example.
4) In LXTerminal, type the following at the $ prompt:

   python improvedisten.py

5) Once again, the cat will confirm that broadcasts are being received from Python, but this time you will also note that there is a *variable* named **SerialNumber** up in the top left of the window with the Scratch cat. This should reflect the serial number of the Pipsta printer (visible on the underside of the Pipsta printer's paper cup when looking through the clear plastic back face.) *It can be seen that variable data(in this case the serial number) can be sent from the Python Listener and placed in a variable in Scratch.* This opens up considerable opportunities for NFC and web interactions with Scratch programs.
6) In Scratch, now try pressing [t],[up arrow], [u], [right arrow] and [down arrow]. Notice that these broadcasts are now producing prints of different types using basic font modes native to the Pipsta printer.
7) Now press [b] to print out the Pipsta printer serial number as a barcode. This demonstrates Scratch's ability to receive data from the Python, process it, and send it back. Using *sensors* and broadcasts in this way opens-up opportunities for NFC and web interactions with Scratch.
8) Now click on the serial number box, and move the slider to any value:

9) Press [b] again to see this variable printed as a barcode. This quickly demonstrates that variables can be passed back and forth as required, allowing Scratch to modify variables automatically by dint of its program, or by Scratch exposing these variable to the user of the Scratch program. This also makes provision for Scratch to delegate significant processing tasks to Python, and for the results to be presented back in Scratch.
10) *Improvedlisten.py* is slightly more sophisticated than its predecessor, but the essential differences are:
    - USB functionality has been introduced to send data to (and –in the case of the serial number—retrieve data from) the printer
    - The *stubs* in the broadcast handlers are now functional and produce printing actions or changes to printer settings when invoked:

```
for msg in listen(scratch_conn):
    if msg[0] == 'broadcast':
        # Handle able command types
        if msg[1] == 'cmd_barcode3of9':
            printing_barcode = True
        elif msg[1] == 'cmd_doubleHeight':
            set_double_height(pr_out)
            printing_barcode = False
        elif msg[1] == 'cmd_doubleWidth':
            set_double_width(pr_out)
            printing_barcode = False
        elif msg[1] == 'cmd_underlined':
            set_underlined(pr_out)
```

```
            printing_barcode = False
        elif msg[1] == 'cmd_normalText':
            set_normal(pr_out)
            printing_barcode = False
        else:
            print_to_ap1400(pr_out, msg[1], printing_barcode)
            printing_barcode = False
```

11) Exit Scratch, noting that the Python script will also terminate at this time.

Certificate.py and numberbonds.sb

The Scratch and Python Listener tutorial concludes with a 'fully-featured' (although basic) Scratch 'Number Bonds' game (applicable to Early Years Foundation Stage and Key Stage 1 of the National Curriculum). At the end of the game, there is a long series of Scratch broadcasts to the Python Listener which result in a certificate being printed.

Broadcasts work in the same way as with previous scripts, and –whilst the handling in the Listener script is more sophisticated (in order to demonstrate optimal or 'Pythonic' coding) previous implementations could simply have been extended to give similar results.

The main differences are in the enhancements to the command set. Previous examples relied on simple print modes native to the Pipsta printer, but this example brings-in powerful Python imaging libraries and graphics commands to improve the perceived worth of the print-out.

Certificate Command Set

At the time of writing, these are:

Scratch Broadcast	Python Listener Handler
cmd_newline	send_newline()
cmd_doubleheight	double_height_text()
cmd_doublewidth	double_width_text()
cmd_doubleheight_and_width	set_double_height_and_width()
cmd_underlined	set_underlined_text()
cmd_normaltext	set_normal_text()
cmd_centre_justify	set_text_hcentred()
cmd_top_flourish	print_top_flourish()
cmd_mid_flourish	print_mid_flourish()
cmd_top_flourish	print_bottom_flourish()
cmd_scratch_image	print_scratch_image()
cmd_gen_name	start_printing_pupils_name()
cmd_display_name	print_pupils_name()
cmd_barcode3of9	start_barcode()
cmd_exit	exit()
<anything else>	process_data()

send_newline()
Issues a single newline. **NB:** Five newlines are required to feed the print-job past the tear-bar.

double_height_text()
Sends a command to the printer to switch the printer's font mode to double-height. See also double_width_text(), double_height_and_width(), set_underlined_text() and set_normal_text()

double_width_text()
Sends a command to the printer to switch the printer's font mode to double-width. See also double_height_text(), double_height_and_width(), set_underlined_text() and set_normal_text()

set_double_height_and_width()
Sends a command to the printer to switch the printer's font mode to both double-height and double-width. See also double_height_text(),double_width_text(),, set_underlined_text() and set_normal_text()

set_underlined_text()
Sends a command to the printer to switch the printer's font mode to underlined. See also double_width_text(), double_height_text(), double_height_and_width() and set_normal_text()

set_normal_text()
Reverts any existing double-height, double-width, combined double-height and double-width and underlined text to normal.

set_text_hcentred()
Subsequent text will be centre justified.

print_top_flourish()
Loads a specific graphic image "*TopFlourish.png*", converts it to a *bitarray* and sends it to the printer as a series of graphics commands.

print_mid_flourish()
Loads a specific graphic image "*MidFlourish.png*", converts it to a *bitarray* and sends it to the printer as a series of graphics commands.

print_bottom_flourish()
Loads a specific graphic image "*TopFlourish.png*", rotates it through 180degrees, converts it to a *bitarray* and sends it to the printer as a series of graphics commands.

print_scratch_image()
Loads a specific graphic image "*Scratch.png*", converts it to a *bitarray* and sends it to the printer as a series of graphics commands.

start_printing_pupils_name()
This function *prepares* an image of the pupil's name in readiness for a subsequent call to *start_printing_pupils_name()*. This allows the image to be generated during the Scratch game and

minimises the delay thereafter. Any text data received after this command will be interpreted as the pupil's name.

print_pupils_name()

This function commences the printing of a ready-prepared image of the pupil's name. This image is generated during game-play by *print_pupils_name().*

start_barcode()

Any text data received after this command will be interpreted as the data to be encoded into a barcode. The barcode *symbology* is set by default to Code3of9.

exit()

Disconnects from Scratch and exits.

process_data()

This is a default handler for anything that is not handled by the above commands. Data received by process_data() will be treated as text, and output directly to the printer as simple text. Note that the printer's font mode will persist, so if –for example—this command is preceded by a call to *double_height_text()* this text will be printed at double height.

Altering the Script

There are many (exciting) avenues for expansion and enhancement of this Python listener. Consider some of the following:

- Add in left and right justification,
- Adding a date/time-stamp to the listener and printing this,
- Developing the 'command set' handled by the look-up table of functions to allow Scratch to define which graphic to print, degrees of rotation etc.
- Extending the scheme of preparing large images during gameplay for a quicker print-job process thereafter,
- Saving/appending the name, result, and review info to a file or MySQL database **on the Raspberry Pi,**
- Encoding results into a QR Code which invokes a webpage and automatically adds a new record to a database of results for the class. ***PIPSTA106 - QR-Code Python Tutorial*** and ***PIPSTA010-PIPSTA014*** and ***PIPSTA011-PIPSTA114*** for guidance on these aspects.
- 'Pushing' NFC data to the Pipsta printer, for results retrieval,
- 'Pulling' NFC data from the Pipsta printer as a sensor value,
- Receiving instructions over the web as sensor values,
- Adjust the command set to provide a Scratch-programmed printer language and deliver:
    - Labels
    - Receipts
    - Menus
    - Merits
    - Play/learning currency

- Measurement logging
- Etc!

We would love your feedback, suggestions and contributions to this particularly fertile area of development, so –if you have a suggestion—please let us know.

Shutting Pipsta Down Safely

Whilst the printer is resilient when it comes to powering down, the Raspberry Pi must undergo a strict shutdown process to avoid corrupting the Micro SD card. The most straightforward method of doing this is to double-click the 'Shutdown' icon on the desktop.

If you are already in LXTerminal, type **sudo shutdown –h now** to shut-down the Raspberry Pi immediately.

Always make sure ALL activity on the Raspberry Pi's green LED (the LED on the right) has stopped before removing the power!

■End of Document■