Quick Reference Tables

Chapter 1

Raspberry Pi Startup Command Quick Reference Table	
Command	**Description**
`sudo`	Gives the user root or super user permissions.
`sudo halt`	Shuts down (halts) the power to the Raspberry Pi.
`sudo reboot`	Shuts down the power to the Raspberry Pi and then restarts it.

Achievement Unlocked: Your Raspberry Pi is up and running!

Chapter 2

Command Line Quick Reference Table	
Command	**Description**
`cat`	Displays the contents (catalog) of the text file.
`cd`	Changes directory. For example, the command cd Desktop moves you into the Desktop directory.
`cd ..`	Moves you up the directory tree to the parent directory.
`cp`	Makes another copy of a file.
`clear`	Allows you to clear the terminal.
`date`	Displays the time and date.
`ls`	Displays a list of files and folders in the current directory.
`ls -l`	Provides a list that includes more detail about the files. The `-l` parameter is a lowercase L (for *long*), not the numeral 1.
`man`	Displays the manual or description file for the command.
`mv`	Moves a file to a new location.
`mkdir`	Makes a directory.
`nano`	Opens the nano text editor. To open a specific text file, add the filename; for example, nano hello opens the hello text file.
`pwd`	Prints the working directory (shows which directory you are currently working in).
`rm xxx`	Deletes (removes) the file named xxx.
`rmdir`	Deletes (removes) a directory.
`sudo`	Gives the user root or super user permissions.
`sudo apt-get install xxx`	Tells the Raspberry Pi to use the Internet to find, download and install the xxx application.
`sudo apt-get update`	Downloads information about any new versions available for applications on your Raspberry Pi.
`sudo apt-get upgrade`	Installs available upgrades for all applications on your Raspberry Pi.
`sudo halt`	Shuts down (halts) the power to the Raspberry Pi.
`sudo reboot`	Shuts down the power to the Raspberry Pi and then restarts it.

Achievement Unlocked: Your Raspberry Pi responds to your commands!

Chapter 3

Scratch Command Quick Reference Table	
Command	**Description**
Control Blocks	
`broadcast x`	Sends a message to all the sprites and the stage which can be used to synchronize scripts across multiple sprites and the stage.
`forever`	Repeatedly iterates actions within set.
`forever if`	Checks whether a condition is true, over and over. If the condition is true the program runs the blocks inside.
`if...else`	If the condition is true, the program runs the blocks inside the if section. If not, it runs the blocks inside the else section.
`repeat x`	Sets number of times for action to repeat.
`stop all`	Stops all scripts for all sprites.
`wait x secs`	Sets time before executing next command.
`when`  `clicked`	Begins script when green flag icon is clicked.
`when I receive x`	Begins script when a set broadcast message is heard.
`When x key pressed`	Begins script when designated key is pressed.
Motion Blocks	
`change x by _`	Changes sprite's position on the stage x axis by a specified amount.
`change y by _`	Changes sprite's position on the stage y axis by a specified amount.
`go to x:_ y:_`	Moves sprite to set x and y coordinates on the stage.
`if on edge, bounce`	Turns sprite in the opposite direction if it touches the edge of the stage.
`move x steps`	Moves sprite forward or backwards *x* number of steps.
`point in direction x`	Points sprite in direction *x*.
`point towards x`	Points sprite towards another sprite or a mouse cursor.
`set x to _`	Sets sprite's position on the stage x axis to a designated place.
`set y to _`	Sets sprite's position on the stage y axis to a designated place.
`turn (clockwise) x degrees`	Rotates sprite clockwise *x* degrees.
`turn (anti-clockwise) x degrees`	Rotates sprite anti-clockwise *x* degrees.

Command	Description
Looks Blocks	
`change size by x`	Changes sprite's size by *x* amount.
`hide`	Hides a sprite from the stage.
`next costume`	Changes sprite's costume to the next costume in the list.
Command	**Description**
`say xxx`	Shows sprite's speech bubble saying *xxx*.
`set size to x`	Sets a sprite's size to *x* percent of its original size.
`show`	Makes a sprite appear on the stage.
`switch to background x`	Changes the background of the stage.
`switch to costume x`	Changes the costume of a sprite.
`think xxx`	Shows sprite's thought bubble thinking *xxx*.
Variables Blocks	
`Change variable by x`	Changes the variable by *x* amount.
`Make a variable`	Creates a new variable that you can name for either a single sprite or for all sprites.
`Set variable to x`	Sets the variable value to *x*.
Sensing Blocks	
`key x pressed`	If *x* key on the keyboard is pressed then reports true.
`touching color x`	If a sprite is touching a designated colour then reports true.
`touching x`	If a sprite is touching designated sprite, edge or mouse cursor, then reports true.

Achievement Unlocked: You have created a program using Scratch!

Chapter 4

Turtle Graphics Command Quick Reference Table	
See also the Scratch Quick Reference Table in Adventure 3	
Commands	**Description**
Pen Blocks (Scratch)	
`change pen color by x`	Changes pen's colour by *x* amount.
`change pen shade by x`	Changes the pen's shade by *x* amount.
`clear`	Clears all pen marks and stamps from the stage.
`pen down`	Puts down a sprite's pen so that it will draw.
`pen up`	Lifts a sprite's pen so it does not draw.
`set pen color to x`	Sets a pen's colour to your choice.
`set pen shade to x`	Sets the pen's shade by *x* amount.
`set pen size x`	Set's a pen's line thickness to *x*.
`stamp`	Stamps a sprite's image on to the stage.
Turtle Module in Python	
`import turtle`	Imports the `turtle` module into Python. Should be at the start of any Python Turtle program.
Creating and Naming the "turtle"	
`alex = turtle.Turtle()`	Opens the Turtle Graphics window, with an arrow cursor in the centre, named *alex*. The arrow cursor represents the turtle, whose movements create your drawing.
Move and Draw	
`forward(x)`	Moves the turtle forward by the specified distance *x*, in the direction the turtle is headed.
`left(x)`	Turns turtle left by *x* units.
`right(x)`	Turns turtle right by *x* units.
`stamp()`	Stamps a copy of the turtle shape onto the canvas at the current turtle position.
Drawing State	
`pendown()`	Puts the pen down and draws when it moves.
`penup()`	Picks the pen up and stops drawing.
`pensize(x)`	Sets the thickness of the line drawn to *x* pixels.
Turtle State	
`shape("turtle")`	Sets the cursor icon. Possible values for the shape are arrow, turtle, circle, square, triangle, classic.
Colour Control	
`color("brown")`	Sets pen colour.

Bonus chapter header 9 Trim size: 7.375 in × 9.1875 in October 24, 2017 7:20 PM

Commands	Description
Additional Commands	
`for`	`for` loops are traditionally used when you have a piece of code that you want to repeat *x* number of times. Example: `for i in [0,1,2,3,4,]`
`for i in range():`	A for loop using the `range()` function that creates a list containing numbers.
`range()`	The `range()` function generates a list of numbers in progression.

Achievement Unlocked: You can create Turtle Graphics on your Raspberry Pi!

Chapter 5

Python Command Quick Reference Table	
Command	**Description**
`#`	The `#` symbol is used at the beginning of a code line to indicate the line is a comment, not part of the program's instructions to the computer.
`\n`	Returns a new line in a string.
`break`	Breaks out of a `for` or `while` loop.
`def`	Allows you to define a function of your creation.
`elif`	Short for 'else if', the `elif` syntax allows you to create multiple conditions that make something happen when they return a value of `true`.
`for`	`for` loops are traditionally used when you have a piece of code which that you want to repeat x number of times.
`if`	Sets a condition which, if true, makes something happen.
`if...else`	Sets a condition which, if true, makes one set of things happen, or if false makes a different set of things happen.
`import`	Imports modules and libraries to add more functionality to your code.
`input()`	A function that asks for user input and converts it into a string.
`inventory = ["Torch", "Pencil", "Rubber Band", "Catapult"]`	An example of a list in Python. Lists can contain values or strings that are separated by commas and encased in square brackets.
`name = value`	An example of a variable.
`print()`	A function that prints anything inside the brackets.
`print(inventory[3])`	An example of using the `print()` function to print item number 3 in the inventory list.
`random`	A Python module that returns a random value.
`return`	The `return` keyword is used when a function is ready to return a value.
`time`	Python module that provides various time-related functions, such as `sleep`.
`while`	A `while` loop continually repeats if a given condition is true.

Achievement Unlocked: You can program in Python on your Raspberry Pi!

Chapter 6

Minecraft Pi Command Quick Reference Table	
Command	**Description**
`from mcpi.minecraft import minecraft`	Imports the Minecraft modules.
`mc = minecraft.create()`	Connects to Minecraft Pi by creating the Minecraft object.
`pos = mc.player.getPos()`	Returns the players position with floats.
`pos = mc.player.getTilePos()`	Returns the players position with integers.
`postToChat(msg)`	Posts a message to chat in Minecraft Pi.
`setBlock`	Sets a block at coordinates.
`setBlocks`	Sets blocks between two sets of coordinates.
`setPos`	Sets the position of a player.

Achievement Unlocked: Why dig when you can code with Minecraft Pi?

Chapter 7

Sonic Pi Command Quick Reference Table	
Command	**Description**
`live_loop :name do...end`	Runs any code between `do` and `end` at the same time as another `live_loop` block.
`play x`	Plays note *x*.
`play_pattern [60,60,67,67,69,69,67]`	Plays a pattern of notes inside a list.
`rand`	Returns a random number.
`.reverse`	An algorithm that reverses the order of notes in a list.
`.shuffle`	An algorithm that shuffles the order of notes in a list.
`use_synth :fm`	Sets the synth sound; in this example, the fm sound.
`with_fx :reverb do ... end`	Adds an effect to a block of sounds; in this example, reverb is added to any code between `do` and `end`.
`x.times do...end`	Runs any code between `do` and `end` *x* number of times.

Achievement Unlocked: Head bopping, toe tapping creator of coded computer music with Sonic Pi!

Chapter 8

GPIO Pins Command Quick Reference Table	
Command	**Description**
`from gpiozero import LED, Button`	Imports the Raspberry Pi gpiozero module
`Led = LED(2)`	Creates an LED against a pin number as well as identifying it as an output
`led.on()`	Sets output GPIO to true or on
`led.off()`	Sets output GPIO to false or off
`if button.is_pressed:`	Sets a condition for a button press
`if pir.motion_ detected:`	Sets a condition for a PIR motion sensor

Achievement Unlocked: Conquering electronics with a Raspberry Pi!

Chapter 9

Accessories Command Quick Reference Table	
Command	**Description**
`From picamera import PiCamera`	Imports the picamera modules
`camera.start_preview()` `camera.stop_preview()`	Starts the camera preview Stops the camera preview
`camera.capture('/home/pi/Pictures/image.jpg')`	Captures a jpeg image and stores it in the Pictures directory
`avconv -r 10 -i image%02d.jpg -qscale 2 timelapse.mp4`	Converts a series of images into a movie using `avconv`
`import explorerhat`	Imports the Explorer HAT module
`explorerhat.light.red.on()` `explorerhat.light.red.off()`	Switches the red LED on the Explorer HAT on and then off
`explorerhat.touch.pressed(button_pressed)`	Posts a message to chat in Minecraft Pi
`from sense_hat import SenseHat` `sense = SenseHat()`	Imports the Sense HAT modules
`sense.show_message("Hello Sense HAT!")`	Scrolls a test string across the LED matrix on the Sense HAT
`temp = sense.get_temperature()`	Gets the current temperature from the temperature sensor on the Sense HAT
`sense.set_pixels(happy)`	Sets all the pixels on the LED matrix of the Sense HAT at the same time

Achievement Unlocked: Experimented with cameras and HATs

Chapter 10

Achievement Unlocked: Your big Raspberry Pi project!