Sparki Learning Command Quick Reference v1.12

(this library makes use of Python3 – versions of Python <3 but >=2.6 *may* work)

General Commands

#

The number sign (often called a hash sign in computer science) indicates the beginning of a comment. Python ignores everything on a line after #. Comments are used to explain what your code is doing and can be very helpful when someone else is reading your code (or you are re-reading it after not having worked on it in some time).

currentTime()

Returns the number of seconds which have occurred since midnight on January 1, 1970. Other functions exist to format this into a more manageable number (like humanTime() below), but this can be used (among other uses) to determine how long a part of a program has taken. For example, to time how long a portion of code takes to execute (in clock time), before the function you could write startingTime = currentTime() and then after the function you could find out the running time with runTime = currentTime() - startingTime

flrange(start, stop, step)

Returns an iterator, very similar to range() in Python 3 or xrange() in Python 2. start should be the first value of x you want to calculate, stop should be the last – step value you want to calculate, and step should be the step. The start, stop, and step values are very similar to the arguments to the range command, but range() allows only integer arguments.

from sparki\_learning import \*

This command imports the sparki\_learning module to make all of the Sparki specific commands available. It must appear at the beginning of your Sparki programs. Other modules (also called libraries) exist in python such as string, math, random, sys, os and datetime.

getName()

Returns the name of the physical Sparki robot. Note that you must have set the name on each physical robot at least once prior to getting the name, or you’ll get a garbage value.

getUptime()

Returns the number of milliseconds since the Sparki was initialized; returns -1 if Sparki has not been initialized.

humanTime()

Returns the time in a human readable format.

init(“[COM?]”)

Connects your computer to the Sparki via Bluetooth. On a Mac, instead of using a COM port, you will use a device path which looks something like “/dev/tty.ArcBotics-DevB”. You must have paired your computer with Sparki on Bluetooth prior to executing this command. Your computer will assign the COM port or device.

randint(LOWER, UPPER)

Returns an integer between (and including) the LOWER and UPPER bounds. To use this function, you must import it from the random library (e.g. from random import \*).

range(MIN, MAX, STEP)

Returns a list of integers from MIN to MAX by STEP. MIN is the number at which the range begins. The range ends one value before MAX. The range counts by STEP. For example, if STEP is 2, range counts by twos. MIN and STEP are optional. MIN defaults to 0 and STEP defaults to 1. (This command is built in to Python.)

setName(NAME)

Sets the name of the physical Sparki robot. Note that you must have set the name on each physical robot at least once prior to getting the name, or you’ll get a garbage value.

wait(TIME)

Waits TIME seconds before moving to the next command.

Common Variables

Several commands in the myro module use variables for SPEED and TIME, including motors(), forward(), backward(), turnLeft(), and turnRight(). wait() makes use only of TIME. MESSAGE is used for output and some input functions. FILE is used for reading data from files and saving data to them.

FILE – a string value containing an absolute path to a file, or a relative path from the directory where IDLE resides. An absolute path on Windows looks like “C:\Music\musicfile.wav” and on Mac looks like “/home/music/musicfile.wav”. Even on Windows, the backslashes may appear as front slashes.

MESSAGE – a value to be output such as a literal string like “Hello, World”, a variable, or a combination. If you want MESSAGE to be a literal string, remember to enclose the string in quotation marks. If you are combining a string plus a numeric value, you need to convert the numeric value to a string using the str() function. For example, if count were a variable which holds the iteration number of a loop, MESSAGE could be “I am on iteration number “ + str(count)

SPEED – a value between -1 and 1. Any value less than -1 will be made -1; any value more than 1 will be made 1. A value of 1 means to turn the wheels at full power. A value of -1 means to turn the wheels in the opposite direction at full power. Decimal values mean to use proportionately less than full power (so .5 is half power).

TIME – the time in seconds to perform the action. Fractional values will perform for fractional seconds. A value of -1 means to perform the action forever. If TIME is optional for a command, the value defaults to -1 (meaning that if you omit TIME, the robot performs the command until you tell it to stop()).

Movement Commands

backward(SPEED, TIME)

Move the robot backward at speed SPEED for TIME seconds. TIME is optional and may be omitted. See the section on Common Variables for an explanation of SPEED and TIME.

forward(SPEED, TIME)

Move the robot forward at speed SPEED for TIME seconds. TIME is optional and may be omitted. See the section on Common Variables for an explanation of SPEED and TIME.

gripperClose(DISTANCE)

Closes the gripper on the robot DISTANCE centimeters. DISTANCE is optional and may be omitted.

gripperOpen(DISTANCE)

Opens the gripper on the robot DISTANCE centimeters. DISTANCE is optional and may be omitted.

gripperStop()

Stops the gripper motor.

motors(LEFT\_SPEED, RIGHT\_SPEED, TIME)

Starts the robot’s wheel motors. The left wheel will move at LEFT\_SPEED. The right wheel will move at RIGHT\_SPEED. The wheels will move for TIME seconds. TIME is optional and may be omitted. LEFT\_SPEED and RIGHT\_SPEED are SPEED variables as defined in the section on Common Variables.

moveBackwardcm(DISTANCE)

Moves the robot backward DISTANCE centimeters.

moveForwardcm(DISTANCE)

Moves the robot backward DISTANCE centimeters.

stop()

Stops the robot’s motors immediately. Only necessary if the robot is presently in motion (e.g. for the motors() or move() commands).

turnBy(VALUE)

Turns the robot by a number of degrees. VALUE can be any number. A negative VALUE turns the robot left (counterclockwise). A positive VALUE turns the robot right (clockwise).

turnTo(VALUE)

Turns the robot to the specified heading relative to the value returned by getAngle(). VALUE can be any number greater than or equal to 0 and less than 360. A value greater than or equal to 360 or less than 0 will be interpreted as a 0.

turnLeft(SPEED, TIME)

Turn the robot left at speed SPEED for TIME seconds. TIME is optional and may be omitted. See the section on Common Variables for an explanation of SPEED and TIME.

turnRight(SPEED, TIME)

Turn the robot right at speed SPEED for TIME seconds. TIME is optional and may be omitted. See the section on Common Variables for an explanation of SPEED and TIME.

Grid Commands

The grid commands implement a pseudo-coordinate plane for use with the Sparki. When you turn the robot on, the robot is assumed to be a 0,0 and facing the positive direction on the y axis. You can use the moveTo() commands to move to a specific point on the grid. Each integer position on the grid is 1cm from the next or previous integer position. For example, 0,1 would be 1cm forward from the starting position of the robot. The robot only updates its grid position when using the grid commands. For example, if you used forward(1,1), that would not update the grid position. If you want to ensure the robot stays somewhere on the grid, only use grid movement commands.

drawFunction(function, xvals, scale = 1)

This is a complicated function. drawFunction() draws the function given by the function argument on the coordinate plane. The function argument should be a lambda function. The lambda function given should return the value of the y coordinate given the x. For example, lambda x: x\*\*2 given as the function would graph y=x2. xvals should be an iterator of the values of x you want to use. You may find the flrange() function helpful. For example, drawFunction( lambda x: math.sin(x), flrange(-2, 2.1, .1) ) would draw the sin x from -2 to 2 going a tenth at a time. scale increases the size of the drawing for visibility.

getAngle()

Return the number of degrees that Sparki has turned using the turnBy() command, or since setAngle() was last called. turnRight(), turnLeft() and motors() do not update the angle.

getPosition()

Return Sparki’s current position on the grid as a list. The x position is the first element of the list (i.e. [0]) and the y position is the second element of the list (i.e. [1]).

moveBy(x, y, turnBack)

Move the robot to grid position x, y as though the current position were 0,0. For example, if the robot is at 3,4, moveBy(1,1) would move Sparki to 4,5. If turnBack is True (not the default), the robot will turn back to the heading it was on prior to the command.

moveTo(x, y, turnBack)

Move the robot to grid position x, y. If turnBack is True (not the default), the robot will turn back to the heading it was on prior to the command.

resetPosition()

Sets Sparki’s current position on the grid to 0,0 and its angle to 0. The same as calling setAngle(0) and setPosition(0,0). Does not move the robot.

setAngle(newAngle)

Sets the number of degrees that Sparki has turned, which is used by moveBy() and moveTo(). When Sparki is initialized, the angle is 0, so to reset the angle as though the robot were just turned on, use setAngle(0). newAngle defaults to 0. turnRight(), turnLeft() and motors() do not update the angle. Does not move Sparki – to turn to an angle relative to the robot’s starting position, use turnTo().

setPosition(x, y)

Sets Sparki’s current position on the grid. Does not move the robot.

Input Commands

Several of the commands (ask(), askQuestion(), joystick(), pickAFile() and yesorno()) will create GUI windows if tkinter is available. If it is not, the library will fall back to pure text.

ask(MESSAGE, TITLE)

Creates window with MESSAGE. User is allowed to input a response. Returns the user’s response. See the section on Common Variables for an explanation of MESSAGE. TITLE is optional and defaults to “Question”.

askQuestion(MESSAGE, OPTIONS, TITLE)

Creates window with MESSAGE, with buttons labeled with OPTIONS. OPTIONS must be a list of strings. Returns the user’s response. See the section on Common Variables for an explanation of MESSAGE. Loops until user gives a value in OPTIONS. TITLE is optional and defaults to “Question”.

gamepad()

Control Sparki using the remote control. The Sparki will not accept commands while it is under remote control. Press + or – on the remote to end remote control.

input(MESSAGE)

Prints MESSAGE to the shell. User is allowed to input a response. Returns the user’s response. See the section on Common Variables for an explanation of MESSAGE. (This command is built in to Python.)

joystick()

Control Sparki using a GUI window. Allows movement and opening and closing of the gripper.

pickAFile()

Creates window with a file dialog so that the user can pick a file. Might be useful for reading from or saving to a file.

yesorno(MESSAGE)

Creates window with MESSAGE, with buttons labeled with “yes” and “no”. Returns the user’s response (which will be either “yes” or “no”). See the section on Common Variables for an explanation of MESSAGE.

Output Commands

beep(TIME, FREQ)

Plays a tone at FREQ for TIME seconds through your computer. Both are optional (FREQ defaults to 2800 and TIME to 200). A list of common frequencies can be found at <http://en.wikipedia.org/wiki/Piano_key_frequencies>.

LCDclear(UPDATE)

Clears (makes blank) Sparki’s LCD. Note that you must call LCDupdate() to display the message once it’s printed if you set UPDATE to False.

LCDprint(MESSAGE, UPDATE)

Prints MESSAGE to the LCD on the back of Sparki. See the section on Common Variables for an explanation of MESSAGE. Note that you must call LCDupdate() to display the message once it’s printed if you set UPDATE to False.

LCDprintLn(MESSAGE, UPDATE)

Prints MESSAGE to the LCD on the back of Sparki, and go to the next line. See the section on Common Variables for an explanation of MESSAGE. Note that you must call LCDupdate() to display the message once it’s printed if you set UPDATE to False.

LCDupdate()

Updates the Sparki’s LCD with anything new you’ve printed to it since the last update.

print(MESSAGE)

Prints MESSAGE to the screen. See the section on Common Variables for an explanation of MESSAGE. print() is built in to Python and has other useful options, but they go beyond what you’re likely to encounter routinely.

setRGBLED(RED, GREEN, BLUE)

Sets the RGB LED to RED, GREEN, BLUE, where each value is a number from 0 to 100. For example, [100, 0, 0] would turn the light fully red, [0, 100, 0] would be fully green, and [0, 0, 100] would be fully blue. The values can be mixed to make most colors.

setStatusLED(BRIGHTNESS)

Sets the status LED to BRIGHTNESS, where BRIGHTNESS is a number from 0 to 100.

Sensor Commands

compass()

Returns Sparki’s current compass heading.

getAccel()

Returns the current values of Sparki’s 3 accelerometers in a list. Accelerometers measure acceleration (primarily due to gravity) and can tell you the orientation of Sparki.

getBattery()

Returns the current voltage of Sparki’s batteries. Anything below around 4 is low. Should be just below 6 with new batteries.

getBright(POSITION)

Returns the brightness value in front of the robot from POSITION. A higher number indicates more light. POSITION is the literal string “left”, “center”, or “right” (or the number 0 for “left”, 1 for “center”, or 2 for “right”). POSITION may be omitted, in which case this returns a list of three values representing the “left”, “center”, and “right” sensors.

getLine(POSITION)

Returns the value of the line sensor in POSITION. The sensor is actually testing the reflection of an infrared light, and is only accurate with a high amount of contrast (e.g. a black line on a white surface). POSITION may be any number 0 through 4: 0 is the line sensor on the left edge, 1 is the left middle, 2 is the middle, 3 is the right middle, and 4 is the left edge. POSITION may be omitted, in which case getLine() returns a list of five values representing all line sensors.

getMag()

Returns the current values of Sparki’s 3 magnetometers in a list. Magnetometers measure magnetic fields around the robot.

getObstacle(POSITION)

Returns a number which is the number of centimeters the Sparki believes the closest object is. POSITION is the literal string “left”, “center”, or “right” (or the number of degrees you want to turn the servo – see the servo command for more information). POSITION may be omitted, in which case getObstacle() returns a list of three values representing the “left”, “center”, and “right” values.

ping()

Returns the number of centimeters to the closest object directly in front of Sparki’s head. You can turn the head with servo(POSITION) and then get a distance with ping().

senses()

Displays a window with (or prints out) data from all of the sensors on Sparki. By default, it updates every two seconds. Program execution is paused while the window is displayed. If tkinter is not available, no window will be displayed, but the status of the sensors will be output in text.

servo(POSITION)

Turns the servo (sparki’s head) to POSITION. POSITION is a number between -90 and 90, where -90 is directly to the left, 0 is straight ahead, and 90 is directly to the right.

Synchronization Commands

The synchronization commands are provided to allow several computers to do something at the same time. In the case of sparki\_learning, they are usually used to allow multiple computers to command their respective robots so that the robots can do things together (at the same time). One computer acts as the server, and all other computers act as the clients. The server is told how much time to wait, and then the server communicates that to any clients that connect to the server. When the time expires, the server and clients all return from the synchronization function.

start\_sync\_server(TIME)

Starts the sync server to execute the commands following after TIME seconds. Opens a network socket on the computer, so you may be asked if you want Python to be allowed to open a port. Defaults to port 32216.

start\_sync\_client(server\_ip, server\_port)

Connects to a sync server and executes the commands following after the period of time specified by the server. server\_ip should be the server's ip address (which is printed to the server's screen when the server calls start\_sync\_server()). server\_port defaults to 32216 (the same as the server).

Conditions

A CONDITION is an expression which evaluates to True or False. True and False are special values in Python called Booleans. The integer 1 can be used for True and 0 can be used for False. A CONDITION is often a comparison. The most common comparisons available in Python are as follows:

|  |  |  |
| --- | --- | --- |
| Comparison | Example | Meaning (i.e. is this statement True or False?) |
| == | x == y | The value of x is equal to the value of y |
| != | x != y | The value of x is not equal to the value of y |
| < | x < y | The value of x is less than the value of y |
| <= | x <= y | The value of x is less than or equal to the value of y |
| > | x > y | The value of x is greater than the value of y |
| >= | x >= y | The value of x is greater than or equal to the value of y |

A CONDITION can also be a combination of comparisons. Comparisons can be combined with and or or.

or is True if **either** of the expressions are True. For example, the expression (1 > 2 or 2 > 1) evaluates to True, because while 1 > 2 is False, 2 > 1 is True.

and is True if **both** of the expressions are True. For example, the expression (1 > 2 and 2 > 1) evaluates to False, because while 2 > 1 is True, 1 > 2 is False.

Comparisons can be chained together with multiple and or or operators. For example, (1 > 2 or 2 > 1 or 3 > 2 and 1 == 1) would evaluate to True.

not reverses a condition – it makes True False and vice versa. For example, not True is False

if statements execute a block of code if a CONDITION is True, and can optionally execute other code if a CONDITION is False. The general format is as follows:

if CONDITION\_1:

code executed if CONDITION\_1 is True

elif CONDITION\_2:

code executed if CONDITION\_1 is False and CONDITION\_2 is True

elif CONDITION\_3:

code executed if CONDITION\_1 and CONDITION\_2 are False and CONDITION\_3 is True

else:

code executed if CONDITION\_1, CONDITION\_2 and CONDITION\_3 are False

The elif and else sections are optional. Often, you will simply have an if clause followed by an else.

Loops

A loop is a section of code which is repeated to iterate through a list or until a condition is met. A loop repeats the body of code until the list is complete or the condition occurs. Python has two kinds of loops: the for loop (which iterates through a list) and the while loop (which continues until a condition is met).

for VARIABLE in LIST:

code inside a loop is indented as part # part of loop

of a block; the loop ends when the code is # part of loop

no longer indented # NOT part of loop

As an example, if you wanted a loop to repeat a series of tasks 4 times, you could write:

for n in range(0,4):

doTask()

LIST can be any Python list. For example, if you wanted to print a series of names, you could write:

for name in (“Dereck”, “Bailey”, “Domonique”, “Hannah”):

print(name)

while CONDITION:

Code inside the loop is continued # part of loop

as long as CONDITION is TRUE. # part of loop

When CONDITION is FALSE and Python # part of loop

reaches the top of the loop # part of loop

the loop ends (or never even begins!) # NOT part of loop

As an example, if you wanted a loop to continue forever, you could write:

while True:

doTask()

You can also use a while loop to perform something for a period of time:

startTime = currentTime()

while ((startTime + TIME) > currentTime()):

doTask()

The CONDITION is described above under the section on Conditions.

Defining a Function

A function is a block of code which performs a particular task. As a general rule, a function should do one thing well. If you find your function is doing multiple things, it might be a good idea to break it down into more functions. A function definition looks something like the following:

def functionName(argumentOne, argumentTwo=defaultValue):

code inside a function is indented as part # part of function

of a block; the function ends when the code is # part of function

no longer indented # NOT part of function

A function can have zero or more arguments (also called parameters). A function can return a value, but is not required to do so. Arguments can be given a default value in the function definition. The argument will have the default value if the caller does not specify another value.

Variable Types

Variables in Python have a type which attaches to the variable upon creation of the variable. Variables in Python do not have to be assigned a type (unlike some other languages).

float

A floating point value – that is, a number with a fractional component like 0.5 or 3.1415.

int

An integer value.

list

A mutable sequence of values. A list is written using square brackets such as [1, 2]. A list can contain variables of different types. Values in the list can be changed (which is what “mutable” means in this context). Individual elements of a list can be addressed by using the name of the list plus square brackets with the number of the item you want to address. For example if you had a list sample\_list = [ 1, “hi”, 3.14 ], you could type sample\_list[0] and that would give you 1 because list numbering starts with position 0. You could type sample\_list[-1] and get 3.14.

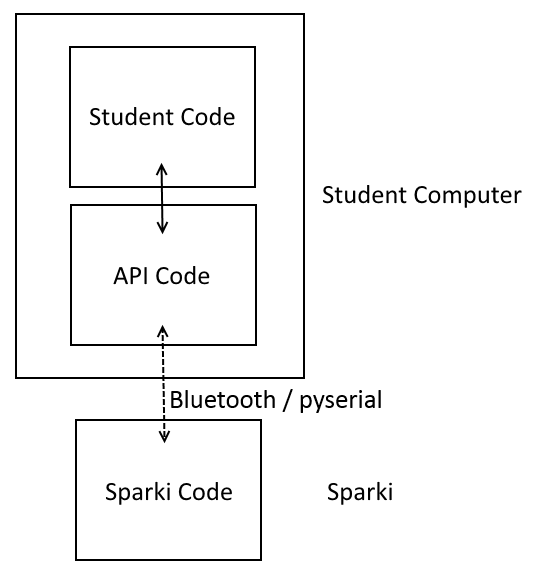
long

An integer value which can be larger than values of type int.

string

A literal string of characters like “Hello, World” or “I love Python.” The string “42” is not the same as the integer 42. Adding two strings together concatenates the strings. For example, “Programming a robot “ + “is easy” would equal “Programming a robot is easy”. Double or single quotation marks are used to delimit a literal string.

Library Design Concept

Sparki Myro consists of two parts: a python library residing on the user’s computer and a Sparki robot which has been loaded with a special Sparki Myro program. The program on the Sparki interprets commands sent by the user’s computer and executes them on the Sparki. Sparki is extremely limited in terms of storage space, and consequently the code which runs on it is simple. Not all features available on Sparki are available in Python – particularly some of the LCD capabilities.