

# Python Overview

## Chapter 1

Start Python.pyw

This is the icon you double-click on to start a Python Shell (IDLE).

```
>>>
```

The Python prompt. This is where you type in a Python command.

Note: All commands you type (including the Myro commands listed above) are essentially Python commands. Later, in this section we will list those commands that are a part of the Python language.

## Chapter 2

```
def <FUNCTION NAME> (<PARAMETERS>) :  
    <SOMETHING>  
    ...  
    <SOMETHING>
```

Defines a new function named `<FUNCTION NAME>`. A function name should always begin with a letter and can be followed by any sequence of letters, numbers, or underscores (`_`), and not contain any spaces. Try to choose names that appropriately describe the function being defined.

## Chapter 3

### Values

Values in Python can be numbers (integers or floating point numbers) or strings. Each type of value can be used in an expression by itself or using a combination

of operations defined for that type (for example, +, -, \*, /, % for numbers). Strings are considered sequences of characters (or letters).

## Names

A name in Python must begin with either an alphabetic letter (a-z or A-Z) or the underscore (i.e. `_`) and can be followed by any sequence of letters, digits, or underscore letters.

```
input(<prompt string>)
```

This function prints out `<prompt string>` in the IDLE window and waits for the user to enter a Python expression. The expression is evaluated and its result is returned as a value of the input function.

```
from myro import *
initialize("comX")

<any other imports>
<function definitions>
def main():
    <do something>
    <do something>
    ...
```

```
main()
```

This is the basic structure of a robot control program in Python. Without the first two lines, it is the basic structure of all Python programs.

```
print <expression1>, <expression2>, ...
```

Prints out the result of all the expressions on the screen (in the IDLE window). Zero or more expressions can be specified. When no expression is specified, it prints out an empty line.

```
<variable name> = <expression>
```

This is how Python assigns values to variables. The value generated by `<expression>` will become the new value of `<variable name>`.

```
range(10)
```

Generates a sequence, a list, of numbers from 0..9. There are other, more general, versions of this function. These are shown below.

```
range(n1, n2)
```

Generates a list of numbers starting from `n1`..`(n2-1)`. For example, `range(5, 10)` will generate the list of numbers `[5, 6, 7, 8, 9]`.

```
range(n1, n2, step)
```

Generates a list of numbers starting from `n1`...(`n2-1`) in steps of `step`. For example, `range(5, 10, 2)` will generate the list of numbers `[5, 7, 9]`.

## Repetition

```
for <variable> in <sequence>:
    <do something>
    <do something>
    ...
```

```
while timeRemaining(<seconds>):
    <do something>
    <do something>
    ...
```

```
while True:
    <do something>
    <do something>
    ...
```

These are different ways of doing repetition in Python. The first version will assign `<variable>` successive values in `<sequence>` and carry out the body once for each such value. The second version will carry out the body for `<seconds>` amount of time. `timeRemaining` is a Myro function (see above). The last version specifies an un-ending repetition.

## Chapter 4

```
True, False
```

These are Boolean or logical values in Python. Python also defines `True` as 1 and `False` as 0 and they can be used interchangeably.

```
<, <=, >, >=, ==, !=
```

These are relational operations in Python. They can be used to compare values. See text for details on these operations.

```
and, or, not
```

These are logical operations. They can be used to combine any expression that yields Boolean values.

```
random()
```

Returns a random number between 0.0 and 1.0. This function is a part of the random library in Python.

`randint(A, B)`

Returns a random number in the range A (inclusive) and B (exclusive). This function is a part of the random library in Python.

## Chapter 5

```
if <CONDITION>:  
    <statement-1>  
    ...  
    <statement-N>
```

If the condition evaluates to True, all the statements are performed. Otherwise, all the statements are skipped.

`return <expression>`

Can be used inside any function to return the result of the function.

`<string>.split()`

Splits `<string>` into a list.

`urlopen(<URL>)`

Establishes a stream connection with the `<URL>`. This function is to be imported from the Python module `urlopen`.

`<stream>.read()`

Reads the entire contents of the `<stream>` as a string.

### Lists:

`[]` is an empty list.

`<list>[i]`

Returns the `i`th element in the `<list>`. Indexing starts from 0.

`<value> in <list>`

Returns True if `<value>` is in the `<list>`, False otherwise.

`<list1> + <list2>`

Concatenates `<list1>` and `<list2>`.

`len(<list>)`

Returns the number of elements in a list.

`range(N)`

Returns a list of numbers from 0..N

```
range(N1, N2, N3)
```

Returns a list of numbers starting from N1 and less than N3 incrementing by N3.

```
<list>.sort()
```

Sorts the <list> in ascending order.

```
<list>.append(<value>)
```

Appends the <value> at the end of <list>.

```
<list>.reverse()
```

Reverses the elements in the list.

## Chapter 6

The if-statement in Python has the following forms:

```
if <condition>:
    <this>

if <condition>:
    <this>
else:
    <that>

if <condition-1>:
    <this>
elif <condition-2>:
    <that>
elif <condition-3>:
    <something else>
...
...
else:
    <other>
```

The conditions can be any expression that results in a True, False, 1, or 0 value. Review Chapter 4 for details on writing conditional expressions.

## Chapter 7

The math library module provides several useful mathematics functions. Some of the commonly used functions are listed below:

**ceil(x)** Returns the ceiling of x as a float, the smallest integer value greater than or equal to x.

**floor(x)** Returns the floor of x as a float, the largest integer value less than or equal to x.

**exp(x)** Returns  $e^{**x}$ .

**log(x[, base])** Returns the logarithm of x to the given base. If the base is not specified, return the natural logarithm of x (i.e., the logarithm to base e).

**log10(x)** Returns the base-10 logarithm of x.

**pow(x, y)** Returns  $x^{**y}$ .

**sqrt(x)** Returns the square root of x.

Trigonometric functions

**acos(x)** Returns the arc cosine of x, in radians.

**asin(x)** Returns the arc sine of x, in radians.

**atan(x)** Returns the arc tangent of x, in radians.

**cos(x)** Returns the cosine of x radians.

**sin(x)** Returns the sine of x radians.

**tan(x)** Returns the tangent of x radians.

**degrees(x)** Converts angle x from radians to degrees.

**radians(x)** Converts angle x from degrees to radians.

The module also defines two mathematical constants:

**pi** The mathematical constant pi.

**e** The mathematical constant e.

## Chapter 8

In this chapter we presented informal *scope rules* for names in Python programs. While these can get fairly complicated, for our purposes you need to know the distinction between a *local name* that is local within the scope of a function versus a *global name* defined outside of the function. The text ordering defines what is accessible.

## Chapter 9 & 10

There were no new Python features introduced in this chapter.

## Chapter 11

The only new Python feature introduced in this chapter was the creation of modules. Every program you create can be used as a library module from which you can import useful facilities.





# Myro

# Overview

Below is a chapter by chapter summary of all the Myro features introduced in this text. For a more comprehensive listing of all the Myro features you should consult the Myro Reference Manual.

## Chapter 1

```
from myro import *
```

This command imports all the robot commands available in the Myro library. We will use this whenever we intend to write programs that use the robot.

```
initialize(<PORT NAME>)
```

```
init(<PORT NAME>)
```

This command establishes a wireless communication connection with the robot. `<PORT NAME>` is determined at the time you configured your software during installation. It is typically the word `com` followed by a number. For example, `"com5"`. The double quotes (") are essential and required.

```
beep(<TIME>, <FREQUENCY>)
```

Makes the robot beep for `<TIME>` seconds at frequency specified by `<FREQUENCY>`.

```
getName()
```

Returns the name of the robot.

```
setName(<NEW NAME>)
```

Sets the name of the robot to `<NEW NAME>`. The new name should be enclosed in double quotes, no spaces, and not more than 16 characters long. For example:

```
setName("Bender").
```

`gamepad()`

Enables manual control of several robot functions and can be used to move the robot around.

## Chapter 2

`backward(SPEED)`

Move backwards at `SPEED` (value in the range -1.0...1.0).

`backward(SPEED, SECONDS)`

Move backwards at `SPEED` (value in the range -1.0...1.0) for a time given in `SECONDS`, then stop.

`forward(SPEED)`

Move forward at `SPEED` (value in the range -1.0..1.0).

`forward(SPEED, TIME)`

Move forward at `SPEED` (value in the range -1.0...1.0) for a time given in seconds, then stop.

`motors(LEFT, RIGHT)`

Turn the left motor at `LEFT` speed and right motor at `RIGHT` speed (value in the range -1.0...1.0).

`move(TRANSLATE, ROTATE)`

Move at the `TRANSLATE` and `ROTATE` speeds (value in the range -1.0...1.0).

`rotate(SPEED)`

Rotates at `SPEED` (value in the range -1.0...1.0). Negative values rotate right (clockwise) and positive values rotate left (counter-clockwise).

`stop()`

Stops the robot.

`translate(SPEED)`

Move in a straight line at `SPEED` (value in the range -1.0...1.0). Negative values specify backward movement and positive values specify forward movement.

`turnLeft(SPEED)`

Turn left at `SPEED` (value in the range -1.0...1.0)

`turnLeft(SPEED, SECONDS)`

Turn left at `SPEED` (value in the range -1.0..1.0) for a time given in seconds, then stops.

```
turnRight (SPEED)
```

Turn right at `SPEED` (value in the range -1.0..1.0)

```
turnRight (SPEED, SECONDS)
```

Turn right at `SPEED` (value in the range -1.0..1.0) for a time given in seconds, then stops.

```
wait (TIME)
```

Pause for the given amount of `TIME` seconds. `TIME` can be a decimal number.

## Chapter 3

```
speak (<something>)
```

The computer converts the text in `<something>` to speech and speaks it out.

`<something>` is also simultaneously printed on the screen. Speech generation is done synchronously. That is, anything following the `speak` command is done only after the entire thing is spoken.

```
speak (<something>, 0)
```

The computer converts the text in `<something>` to speech and speaks it out.

`<something>` is also simultaneously printed on the screen. Speech generation is done asynchronously. That is, execution of subsequent commands can be done prior to the text being spoken.

```
timeRemaining (<seconds>)
```

This is used to specify timed repetitions in a while-loop (see below).

## Chapter 4

```
randomNumber ()
```

Returns a random number in the range 0.0 and 1.0. This is an alternative Myro function that works just like the `random` function from the Python `random` library (see below).

```
askQuestion (MESSAGE-STRING)
```

A dialog window with `MESSAGE-STRING` is displayed with choices: 'Yes' and 'No'. Returns 'Yes' or 'No' depending on what the user selects.

```
askQuestion (MESSAGE-STRING, LIST-OF-OPTIONS)
```

A dialog window with `MESSAGE-STRING` is displayed with choices indicated in `LIST-OF-OPTIONS`. Returns option string depending on what the user selects.

```
currentTime ()
```

The current time, in seconds from an arbitrary starting point in time, many years ago.

`getStall()`

Returns `True` if the robot is stalled when trying to move, `False` otherwise.

`getBattery()`

Returns the current battery power level (in volts). It can be a number between 0 and 9 with 0 indicating no power and 9 being the highest. There are also LED power indicators present on the robot. The robot behavior becomes erratic when batteries run low. It is then time to replace all batteries.

## Chapter 5

`getBright()`

Returns a list containing the three values of all light sensors.

`getBright(<POSITION>)`

Returns the current value in the `<POSITION>` light sensor. `<POSITION>` can either be one of 'left', 'center', 'right' or one of the numbers 0, 1, 2.

`getGamepad(<device>)`

`getGamepadNow(<device>)`

Returns the values indicating the status of the specified `<device>`. `<device>` can be "axis" or "button". The `getGamepad` function waits for an event before returning values. `getGamepadNow` immediately returns the current status of the device.

`getIR()`

Returns a list containing the two values of all IR sensors.

`getIR(<POSITION>)`

Returns the current value in the `<POSITION>` IR sensor. `<POSITION>` can either be one of 'left' or 'right' or one of the numbers 0, 1.

`getLight()`

Returns a list containing the three values of all light sensors.

`getLight(<POSITION>)`

Returns the current value in the `<POSITION>` light sensor. `<POSITION>` can either be one of 'left', 'center', 'right' or one of the numbers 0, 1, 2. The positions 0, 1, and 2 correspond to the left, center, and right sensors.

`getObstacle()`

Returns a list containing the values of all IR sensors.

```
getObstacle (<POSITION>)
```

Returns the current value in the <POSITION> IR sensor. <POSITION> can either be one of 'left', 'center', or 'right' or one of the numbers 0, 1, or 2.

```
savePicture (<picture>, <file>)
```

```
savePicture ([<picture1>, <picture2>, ...], <file>)
```

Saves the picture in the file specified. The extension of the file should be ".gif" or ".jpg". If the first parameter is a list of pictures, the file name should have an extension ".gif" and an animated GIF file is created using the pictures provided.

```
senses ()
```

Displays Scribbler's sensor values in a window. The display is updated every second.

```
show (<picture>)
```

Displays the picture in a window. You can click the left mouse anywhere in the window to display the (x, y) and (r, g, b) values of the point in the window's status bar.

```
takePicture ()
```

```
takePicture ("color")
```

```
takePicture ("gray")
```

Takes a picture and returns a picture object. When no parameters are specified, the picture is in color.

## Chapter 6 & 7

No new Myro features were introduced in these chapters.

## Chapter 8

```
GraphWin ()
```

```
GraphWin (<title>, <width>, <height>)
```

Returns a graphics window object. It creates a graphics window with title, <title> and dimensions <width> x <height>. If no parameters are specified, the window created is 200x200 pixels.

```
<window>.close ()
```

Closes the displayed graphics window <window>.

```
<window>.setBackground (<color>)
```

Sets the background color of the window to be the specified color. <color> can

be a named color (Google: color names list), or a new color created using the `color_rgb` command (see below)

```
color_rgb(<red>, <green>, <blue>)
```

Creates a new color using the specified `<red>`, `<green>`, and `<blue>` values. The values can be in the range 0..255.

```
Point(<x>, <y>)
```

Creates a point object at (`<x>`, `<y>`) location in the window.

```
<point>.getX()
```

```
<point>.getY()
```

Returns the x and y coordinates of the point object `<point>`.

```
Line(<start point>, <end point>)
```

Creates a line object starting at `<start point>` and ending at `<end point>`.

```
Circle(<center point>, <radius>)
```

Creates a circle object centered at `<center point>` with radius `<radius>` pixels.

```
Rectangle(<point1>, <point2>)
```

Creates a rectangle object with opposite corners located at `<point1>` and `<point2>`.

```
Oval(<point1>, <point2>)
```

Creates an oval object in the bounding box defined by the corner points `<point1>` and `<point2>`.

```
Polygon(<point1>, <point2>, <point3>, ...)
```

```
Polygon([<point1>, <point2>, ...])
```

Creates a polygon with the given points as vertices.

```
Text(<anchor point>, <string>)
```

Creates a text anchored (bottom-left corner of text) at `<anchor point>`. The text itself is defined by `<string>`.

```
Image(<centerPoint>, <file name>)
```

Creates an image centered at `<center point>` from the image file `<file name>`. The image can be in GIF, JPEG, or PNG format.

All of the graphics objects respond to the following commands:

```
<object>.draw(<window>)
```

Draws the `<object>` in the specified graphics window `<window>`.

```
<object>.undraw()
```

**Undraws** <object>.

```
<object>.getCenter()
```

**Returns the center point of the** <object>.

```
<object>.setOutline(<color>)
```

```
<object>.setFill(<color>)
```

**Sets the outline and the fill color of the** <object> **to the specified** <color>.

```
<object>.setWidth(<pixels>)
```

**Sets the thickness of the outline of the** <object> **to** <pixels>.

```
<object>.move(<dx>, <dy>)
```

**Moves the object** <dx>, <dy> **from its current position.**

The following sound-related functions were presented in this chapter.

```
beep(<seconds>, <frequency>)
```

```
beep(<seconds>, <f1>, <f2>)
```

**Makes the robot beep for** <seconds> **time at frequency specified. You can either specify a single frequency** <frequency> **or a mix of two:** <f1> **and** <f2>.

```
<robot/computer object>.beep(<seconds>, <frequency>)
```

```
<robot/computer object>.beep(<seconds>, <f1>, <f2>)
```

**Makes the robot or computer beep for** <seconds> **time at frequency specified. You can either specify a single frequency** <frequency> **or a mix of two:** <f1> **and** <f2>.

```
robot.playSong(<song>)
```

**Plays the** <song> **on the robot.**

```
readSong(<filename>)
```

**Reads a song file from** <filename>.

```
song2text(song)
```

**Converts a** <song> **to text format.**

```
makeSong(<text>)
```

```
text2song(<text>)
```

**Converts** <text> **to a song format.**

## Chapter 9

```
getHeight(<picture>)  
getWidth(<picture>)
```

Returns the height and width of the <picture> object (in pixels).

```
getPixel(<picture>, x, y)
```

Returns the pixel object at *x*, *y* in the <picture>.

```
getPixels(<picture>)
```

When used in a loop, returns one pixel at a time from <picture>.

```
getRGB(pixel)  
getRed(<pixel>)  
getGreen(<pixel>)  
getBlue(<pixel>)
```

Returns the RGB values of the <pixel>.

```
makeColor(<red>, <green>, <blue>)
```

Creates a color object with the given <red>, <green>, and <blue> values (all of which are in the range [0..255]).

```
makePicture(<file>)  
makePicture(<width>, <height>)  
makePicture(<width>, <height>, <color>)
```

Creates a picture object either by reading a picture from a <file>, or of the given <width> and <height>. If <color> is not specified, the picture created has a white background.

```
pickAColor()
```

Creates an interactive dialog window to select a color visually. Returns the color object corresponding to the selected color.

```
pickAFile()
```

Creates an interactive dialog window that allows user to navigate to a folder and select a file to open. Note: it cannot be used to create new files.

```
repaint()  
repaint(<picture>)
```

Refreshes the displayed <picture>.

```
savePicture(<picture>, <file>)  
savePicture(<picture list>, <gif file>)
```

Saves the <picture> in the specified file (a GIF or JPEG as determined by the



extension of the <file>: .gif or .jpg). <picture list> is saved as an animated GIF file.

```
setColor(<pixel>, <color>)  
setRed(<pixel>, <value>)  
setGreen(<pixel>, <value>)  
setBlue(<Pixel>, <value>)
```

Sets the color of <pixel> to specified <color> or <value>.

```
show(<picture>)  
show(<picture>, <name>)
```

Displays the <picture> on the screen in a window named <name> (string).

```
takePicture()  
takePicture("gray")  
takePicture("blob")
```

Takes a picture from the Scribbler camera. It is a color picture by default, or grayscale ("gray"), or a filtered image based on the defined blob ("blob"). See chapter text for examples.

## **Chapter 10**

There were no new Myro features introduced in this chapter. Actually, when the chapter is complete it will have Myro primitives for neural nets/conx described here.

## **Chapter 11 & 12**

No new Myro features were introduced in this chapter.

# Index

## A

AAAI 15  
ABC song 56  
abstraction 29  
acos **153**, 272  
Aesop 35  
Aggressive, Vehicle#2 112-13  
Aibo 4  
algorithm 233, **243**, 252-53  
Al Khwarizmi 235  
Alive, vehicle#1 109-12  
Anananova 263  
and 71, 72, **76**, 269  
Animated GIF 185-86  
AntiqueWhite 159  
append 93, **104**, 271  
Apple Strudel 233  
Artificial Intelligence 4, 73, **205-225**  
    movie 3, 205  
Asimov's Laws 14  
asin **153**, 272  
askQuestion 65-66, **75**, 114, 277  
assignment 41, **53**, 268  
atan **153**, 272  
Avoiding obstacles 123-25

## B

backward 19, **31**, 276  
battery disposal 73  
beep 10, **12**, 169-71, **177**, 275, 281  
Behavior-based control 132  
Blobs 198  
blob filtering 198  
Bluetooth 5, 7  
Bluetooth logo 15, 33  
blur image 193  
Boolean 71  
boundary conditions 247  
Braitenberg, Valentino 108, 127  
Braitenberg Vehicles 108-

Breazeal, Cynthia 15, 263  
Brooks, Rodney 257-58, 263  
Burglar Alarm 120

## C

C 5  
C++ 244  
CamelCase 56  
case sensitive 39  
ceil 137, **152**, 271  
Celsius 55  
characters 42  
Chazelle, Brian 255  
Chess 209  
Chicken Kabobs 230  
Chinese Room 224  
Ching-Chong-Cha 148  
choice 215, **224**  
Circle 161, **176**, 280  
Clash, The 107  
close 158, **175**, 279  
Cockroach 127  
color\_rgb 162, **175**, 280  
CommonLisp 244  
comments 26  
computable 253  
Computational Linguistics 207  
Computer Science 227-28  
conditions 69-72  
constant time algorithms 252  
Corral Exiting 125  
cos **153**, 272  
Coward, Vehicle#2 112-13  
currentTime 67-69, **75**, 277

## D

Dar es Salaam 92, 93  
data 251  
Data Fountain 264  
dead reckoning 60

decision making 147  
Deep Blue 209  
def 22, **32**, 98, 267  
degrees **153**, 272  
Delaware River 4  
Dijkstra, Edsger W. 227-28  
Direct Control 133  
draw 159, **176**, 280

## E

e **153**, 272  
edge detection 199  
Elvis 70  
emboss image 194-95  
Energy Problem 45  
enlarge image 191-92  
equal temperament 171  
Ericsson 15  
Euclid v  
Error Checking 247-  
Euro 55  
exp 116, 137, **152**, 272  
Explorer 36  
Explorer, Vehicle#3 115  
expression 41

## F

Face Bank 259  
False 52, 69, 70, **76**, 269  
Fahrenheit 55  
Firefox 36  
floating point numbers 41-42  
floor 137, **152**, 272  
Flip Flap 259  
Flops 253  
Fluke Dongle 8, 104  
Flying Circus 23  
Flynn, Anita M. 257-58  
Follower 121  
for 49, **54**, 269  
formal languages 11  
forward 19, **31**, 276  
fractals 178

Franke, Uli 261  
Franklin Institute 97  
from 26, 27, 138-39, 268  
function 22, 32

## G

Gamepad controller 12, 95-96  
gamepad 12, **13**, 98, 276  
Game playing 209-  
Geier, Sven 157  
getBattery 74, **75**, 278  
getBlob 199, **200**  
getBlue **201**, 282  
getBright 87-88, **101-02**, 278  
getGamepad 95-96, **102**, 278  
getGamepadNow 95-96, **102**, 278  
getCenter 162, **176**, 281  
getGreen **201**, 282  
getHeight 183, **200**, 282  
getIR 189, **102**, 278  
getLight 86, **102**, 278  
getName 11, **13**, 275  
getObstacle 90, **102**, 278-79  
getP1 160, **176**  
getP2 160, **176**  
getPixel 187, **201**, 282  
getPixels 188, **201**, 282  
getRed **201**, 282  
getRGB 187, **201**, 282  
getStall 72, **75**, 278  
getX 160, **176**, 280  
getY 160, **176**, 280  
getWidth 183, **200**, 282  
GIF 183  
global name 164-65  
Google 4  
Gore, Al 55  
Gormson, Harald B. 15  
GPS 261-62  
Grade A eggs 234  
GraphWin 156, **175**, 279  
grayscale images 85

Gregorian Calendar 242  
Grey Poupon 38, 94

## H

Hallway Cruiser 120  
Hektor robot 261  
Hertz (Hz) 169  
hi-fidelity 170  
HiLo game 154  
Hoare, C. A. R. 227  
Hogg, David 127  
Hugs & Kisses 210  
Human-robot interaction 262-63

## I

iCat robot 263  
IDLE 8, 22, 23, 29, 38  
Idle, Eric 23  
if-statement 100, **103**, 118, **128**. 270-71  
image 182  
Image 168, **176**, 280  
image processing 190  
image understanding 195  
Imitation Game 206  
import 138-39, 275  
in 92, **103**, 270  
Indecisive 117  
init **13**, 275  
initialize 9, 10, **13**, 275  
input 44-46, **53**, 94, 268  
integers 41  
internet 5  
interoceptors 60  
invocation, function 23  
iPhone 70  
iRobot 2, 3, 39

## J

jalapeno 60  
Jankenpon 148  
Java 5  
Joe, Gigolo 205  
Joel, Billy 157

Jones, Crispin 259  
Jones, Mick 107  
JPEG 183  
Julia Sets 178

## K

Kasparov, Gary 209  
Kismet robot 263  
Kitaoka, Akiyoshi 181  
Koch Snowflakes 178  
Konane 209

## L

Ladybug 107  
Larson, Doug 227  
LavenderBlush 159  
Law, Jude 205  
Leap Frog 260  
leap year 241-  
LED 73  
LEGO Mindstorms 4  
len 92, **104**, 270  
Lenhi, Jurg 261  
Light following 121-22  
Line 160, **176**, 280  
linear time algorithms 252  
List comprehensions 214, **224**  
lists 49, 91-93, 270  
Loan calculator 139-47  
local name 164-65, 273  
localtime **77-78**  
log 137, **152**, 272  
log10 137, **152**, 272  
logarithmic time algorithms 252  
logical operations 71  
loop 49  
loop index variable 49  
Love, Vehicle#3 115  
Lousanne 261

## M

main 36, **53**, 267  
makeColor 187, 189, **201**, 282

`makePicture` 184, 186-, **201**, 282  
 Mandelbrot Sets 178  
 Mars Rover 1, 2, 4  
 Martin, Fred 127  
`math` library 116, 137-38, 271-72  
 Maze solver 125  
 meaning of life 233  
 Measuring Device 121  
 Media Player 36  
 Megapixel 183  
 Mignot, Charles 264  
 Minimax algorithm 219  
 Minsvoort, Koert van 264  
 MIT Media Lab 127, 264  
 mixed case 56  
 module 25-28, **249**  
 Monty Python 5, 23  
 Morris, Errol 258  
 Moscow 92-93  
`motors` 19, **31**, 276  
`move` (robot) 20, **31**, 276  
`move` (graphics object) 166, **177**, 281  
 musical scale 171  
 Myers, Mike 131  
 Myro 5-7, 9, 275  
 myro Song Format 173

## N

Names 39, **53**, 136, 164-66, 268  
 NASA JPL 1, 2, 258  
 Nash, Johnny 81  
 Natural Language Understanding 206  
 natural languages 11, 206  
 Naughts & Crosses 210  
 negative image 194  
 New York 70, 92-93  
 Nexi robot 264  
 Nike vii  
`not` 71, 72, **76**, 269  
 notes 171  
 numbers 41

## O

octave 171  
 OK Corral 25  
 OLPC Project 18, 226-28  
 Opportunity robot 1, 2, 14, 258  
`or` 71, 72, **76**, 269  
 Orb Swarms 257, 265  
 Osment, Haley Joel 59, 205-6  
 Oval **176**, 280

## P

Papal Bull 241-  
 Paper Scissors Rock 147-51, 221-  
 parameters 24, 164-66  
 Paranoid 117  
 Paris 70  
 Paro robot 19  
 Pathfinder 14  
 PB&J 275  
 Pennsylvania 4  
 Philadelphia 97  
`pi` **153**, 272  
`pickAColor` 187, **201**, 282  
`pickAFile` **201**, 282  
 pixels 85, 182-83  
`playSong` 174, **177**, 281  
 Pleo 4, 16-18  
 PNG 183  
`Point` 159, **175**, 280  
 Polar coordinates 202  
 Polka 173  
 Polygon **176**, 280  
 polynomial time algorithms 252  
 Pope Gregory XIII 242  
`pow` 137, **152**, 272  
 Powers, Austin 131  
`print` 37-38, **53**, 268  
 Programming 5, 229  
 Programming Language 5, 36, 232, **243**  
 Proprioception 60  
 proximity sensor 61  
 Python 5, 36, 38  
 Python Shell 5, 9

## Q

quadratic algorithms 252

## R

radians **153**, 272

random 63-64, **76**, 77, 148, 269

randomNumber **75**, 77, 277

randint 64, **76**, 270

range 49, **54**, **93-94**, 268, 270

Reactive behaviors 121-25

Reactive control 133

read **98**, **103**, 270

readSong 174, **177**, 281

Rectangle **176**, 280

Refrigerator Detective 120

repaint 187, **201**, 282

repetition 49, **54**, 147

Resnick, Mitchel 127

return 99-100, **103**, 270

return values 136

reverse 93, **104**, 271

RGB 85, 162, 182, 187

RoboCup 262

Robot, definition 3

Robot Hall of Fame 15

Robot Vision 195

Rochambeau 148

Rock Paper Scissors 147-51, 221-

Roomba 2, 3, 18, 48

rotate 20, **31**, 276

Rotating Snakes 180

Royal Mail 5

Run Module 38

runic alphabet 15

## S

Saab 139

savePicture **84**,**102**,183-,**201**,279,282

scale 171

Scassellati, Brian 263

Science Magazine 127

scope 164

Scribbler 6, 18

Scribbler 174

Scribbler drawings 261

Scribbler sensors 82

Sear, Cole 59, 60

Searle, John 224

sequential execution 147

senses 83, **102**, 279

Sensor Fusion 133

setBackground 159, **175**, 279

setBlue **201**, 283

setColor 187, **201**, 283

setFill 162, **177**, 281

setGreen **202**, 283

setName 11, **13**, 275

setOutline 162, **177**, 281

setPixel 189

setRed **201**, 283

setRGB 188

setWidth 162, **177**, 281

sharpen image 193

shrink image 191-92

shrinking factor 191-92

show 83, **102**, 182, 185, **202**, 279, 283

Shyamalan, M. Night 59, 60

Sierpinski Triangles 178

Simpson, Homer 34-35, 251

sin **153**, 272

Sixth Sense movie 59-60

Snicker's moment 60

Sojourner 14, 258

solvable 253

song2text **177**, 281

SONY 4

sort 93, **104**, 271

Social Robotics 262

Soviet Union 235

Space Complexity 251-

speak 38, **52**, 104, 277

Spielberg, Steven 3, 255

Spirit robot 1, 2, 14, 258

Spitler, Phil 257

split 94, **103**, 270  
 sqrt 137, **152**, 272  
 Squyres, Steve 1  
 Start Python 8, **13**, 267  
 strings 42  
 stop 21, **31**, 276  
 Subsumption Architecture 134  
 Sullivan, Jon 59, 107  
 syntax error 29

**T**

Tag reading pen 260  
 takePicture 83, **103**, 185-,**202**,279,283  
 Takada 260  
 tan **153**, 272  
 Tengu 259-60  
 Testing 247-  
 Text 168, **176**, 280  
 Thingamapoops 260  
 Tic Tac Toe 210-  
 time **77**  
 timeRemaining 51, **54**, 277  
 Timid 117  
 TOMY Company 260  
 Toyota Prius 92  
 Traffic Lights 131  
 translate 20, **31**, 276  
 True 52, 69, **76**, 269  
 Tumbleweed robot 2  
 Turing, Alan 206  
 Turing Test 206  
 turnLeft **31**, 276  
 turnRight 19, **31-32**, 277

**U**

UGOBE Inc. 17  
 uncomputable 253  
 undraw **176**, 281  
 Unicode 70

Unimation 4  
 unsolvable 253  
 urllib 97-98  
 urlopen 97-98, **103**, 270  
 USDA 234

**V**

Values 40, **53**, 267  
 variable 41  
 Victoria Crater, Mars 2

**W**

wait 24, **32**, 277  
 Wales 48  
 Wall Detector 120  
 Washington DC 251  
 Washington state 251  
 while 51, **54**, 68, 72, 269  
 WhiteSmoke 159  
 Wikipedia 15, 23  
 Wong, Yingshun 131  
 world population 43, 51-52, 74  
 world wide web 4

**X**

XO Laptop 226-27

**Y**

Y2K Problem 255

**Z**

Zamboni 47, 76  
 Zefrank 261

## Scribbler: Myro Reference

