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 indication 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 sesnors.

`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 sesnors.

`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

Scribbler: Myro Reference