Introduction to Robotics with the Raspberry Pi

INTRODUCTION TO PYTHON

Part 2

Jeff Cicolani
Saturday, January 22, 2017

This is the second installment of the Introduction to Python Workshop. Like before, his workshop can be done on the Pi, if you have it set up in a configuration that works for you, or you can follow along on a laptop. For the most part the OS isn't going to matter with the exception of how you save and access files. I will be using a laptop with Windows installed, but, since I am using the standard tools that comes with your install of Python, the OS should not matter.

Introduction

The purpose of this series of workshops is to challenge you to build a simple robot that will be expanded over time. It's intended to be difficult. It is intended to provide a hands-on experience to help you past the most difficult part of learning robotics; intimidation by the technology. We're going to teach you some of the basics of robotics the same way I learned how to swim… by being thrown in to the deep end while someone more experienced watches over to make sure you don't just drown.

So, with that, we expect you to take what you experience in these workshops and add to it through your own learning. We're going to get you going in the right direction, but there's not going to be a lot of hand holding. You will have to fill in the gaps, learn the details on your own.

This continuation of an introduction to Python will be no different. Last time we showed you how to install the tools, use the editor, and write some simple programs. We discussed program structure, key syntax and formatting issues, data types, variables, and right into control structures and some of the object-oriented aspects of Python.

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

In this week's workshop, we will be talking more about functions, libraries, and classes. These elements are crucial to making efficient programs that do more than one thing.

At the end of the workshop we don't expect you to be able to write your own programs. What we do expect is for you to know how, and be comfortable with, using the editor, writing code, compiling and executing programs. Most importantly, you should be able to look at someone else's code and be able to read it, have a basic understanding of what they are trying to do, and to identify the building blocks. Dissecting other peoples' code is key to learning quickly.

In term of resources, here's a couple pieces of advice:

1. Community support for Python is excellent. The Python web site is an invaluable source for learning and growing in Python. In particular be sure to check out the beginners page at https://www.python.org/about/gettingstarted/. We will, actually, be starting here for installation in the next section.
2. Get yourself a good book on learning Python. I am fond of Python Crash Course from No Starch Press. I picked mine up at my local book seller, but you can also find it on Amazon (https://www.amazon.com/Python-Crash-Course-Hands-Project-Based/dp/1593276036/ref=sr_1_1?ie=UTF8&qid=1483848620&sr=8-1&keywords=python+crash+course) or Barnes and Nobles (http://www.barnesandnoble.com/w/python-crash-course-eric-matthes/1120977249?ean=9781593276034).

A Brief Review

Then Zen of Python

Tim Peters, a long-time contributor to Python, wrote down the governing principals behind Python development. I think it actually applies to all code and just about everything we do in robotics and, perhaps, in life. They are actually tucked away in an Easter Egg in the Python IDE.

1. From the Start menu, open IDLE (Pyhon/IDLE)
   This will open the IDLE Shell. We'll go over what this is in a moment.
2. Type:

```
import this
```

3. Press enter
   It should display the following (but do it anyway):

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Native Python Tools

The Python Shell

The Python Shell is an interface to the Python interpreter. Technically, if you're a fan of command line, you could launch a terminal window and invoke the Python interpreter in that. If that's how you want to proceed, you'll need to look up more information on configuring Python. In theory the setup program is supposed to take care of that configuration, but I have not had any luck with that and troubleshooting command line interaction is beyond the scope of this workshop. So, instead, we'll be using the Python Shell interface that is installed with Python.

Since we'll be using this interface a lot throughout this series of workshops it would be prudent to learn more about the IDLE interface and the many tools it provides. You can start at the IDLE documentation page at https://docs.python.org/3/library/idle.html.

The Python Text Editor

IDLE has another very important aspect; the text editor. We will be using this throughout these workshops to write our programs and modules. The text editor is another aspect of IDLE and not a separate program, though it will always open in a separate window. You can write Python programs in any text editor and there are a lot of IDEs that also support Python. However, as mentioned in the previous section, the IDLE interface provides a lot advantages.

Running a Program

From IDLE you can simply press F5. You'll need to make sure the file is saved first, but this will run your file. It is the same as selecting **Run → Run Module** from the menu bar.

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

If your system us properly configured to run Python from the command line, you can execute a program from there, as well. You'll need to either navigate to the location of the file or have the full file location in the call. To execute a Python script from the command line you will type `python` followed by the file to be run.

Structure

Indentation

Python uses indentation to indicate a block of code. Blocks of code will be indented to indicate that they are, in fact, a block. If a line in a block is not properly indented, you'll get an error. This is one of the key reasons we'll be using IDLE. It automatically manages indentations. That doesn't mean you, as a user, can't screw this up, it just means this type of error is greatly reduced.

Comments

A comment in Python is any line preceded with a `#`. Python will ignore everything following the "#" to the end of the line.

```
# create a variable to hold the text
message = "Hello World"

# print the text stored in the variable
print(message)
```

A good habit to get into is simply outlining your code with comments before you actually write it. Take some time before you start writing to think about what you need your code to do and how you will go about doing it. Create a flow chart or simply write down the steps you will take to accomplish your goal. Then translate this into a series of comments in your file before you write any actual code. This will help you structure the problem in your mind and improve your overall flow. If you identify a step you are repeating, you likely have a candidate for a function, which we will discuss later.

Comparators

Here is a list of comparators:

Equals	**==**
Not equal	!=
Less than	<
Greater than	>
Less than or equal to	<=
Greater than or equal to	>=

Variables and Data Types

Variables

Variables are essentially a convenient container for storing information. In Python variables are very flexible. You don't need to declare a type. The type is generally determined when you assign a value to it. In fact, you

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

declare a variable by assigning a value to it. This is important to remember when you start with numbers. There is a difference in Python between 1 and 1.0. The first is an integer and the second is a float. More on that shortly.

There are some general rules for variables:

- They can only contain letters, numbers, and underscores
- They are case sensitive. “variable” is not the same as “Variable”. That’s going to bite you later.
- Don’t use Python keywords

In addition to these hard and fast rules, here are a few tips:

- Make the variable name meaningful with as few characters as possible
- Be careful using lower case “L” and upper case “O”. These look very similar to 1 and 0 and can lead to confusion. I’m not saying done use them, just make sure it’s clear what you’re doing.

Data Types

Strings

Stings are a collection of one or more characters contained in quotes. Quotes are how you indicate a string. For instance, “100” is a string, 100 is a number (an integer to be accurate). You can use double or single quotation marks, though remember what you used. You can nest one type of quotes in another. This allows you to use apostrophes in your text.

There are a lot of ways to manipulate strings. We strongly suggest you spend some time exploring a lot of these methods. For more information go to https://docs.python.org/3/tutorial/introduction.html#strings.

Numbers

Numbers in Python come in a couple flavors; Integers and Floats. Integers are whole numbers whereas floats are decimals. If you perform arithmetic with one type, the result will be that type. Math with integers will result in an integer or float. Math with floats will result in a float. If you perform arithmetic with both types, the result will be a float.

Lists

A list is a collection of items in a particular order. In other languages, they are generally known as arrays. You can put anything you want in a list and they don’t even have to be of the same data type. However, if you’re going to be mixing data types in a list, be sure you know what type you are getting out when you go to use it.

You group a list using square brackets []. Lists are 0 based. This means the first element in a list is at position 0, the second is at position 1, etc. You can access the individual elements of a list by calling it’s index, or location within the list.

Lists are very powerful and a very important aspect of Python. Spend some time exploring lists at https://docs.python.org/3/tutorial/introduction.html#lists

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

Tuples

You're going to hear the term "tuple" a lot when working with Python. A tuple is simply a special kind of list that cannot be changed. Think of a tuple as a list of constants, or a constant list. You declare a tuple using parenthesis rather than square brackets.

Dictionaries

A dictionary is similar to a list except that it allows you to name your items in the list. This is done using **key:value** pairs. The key becomes the index for the value. This allows you to add some meaningful structure to your lists. A dictionary is declared using curly brackets rather than square brackets

Control Structures

If Statements

The if statement allows you to test for a condition before you execute a block of code. This next piece of code loops through the robots list and determines whether or not the robot is Nomad.

Loops

Loops allow you to repeat a block of code in order to perform the tasks multiple times. There are two flavors of loop; the for loop and the while loop.

For Loop

A for loop performs a block of code for each element in list.

While Loop

While a for loop will execute a block of code for each element in a list, the while loop will execute a block of code as long as its condition evaluates to true. This is often used to execute code a specific number of times or until while the system is in a specific state. It's also used as a type of main loop, executing the code endlessly. This is called an open loop since there is no close to it. If this happens accidentally, and it will, press `ctrl+c` to exit the program.

Functions

Functions are predefined blocks of code that we can call from within the program to perform a task. For more on the functions available, check out the Python Standard Library:
https://docs.python.org/3/library/index.html

Defining a Function

To define your own function, you will create the name of the function and a block of code that contains the operations you want to perform.

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

```
>>> def hello_world():
        message = "Hello World"
        print(message)


>>> hello_world()
Hello World
```

Passing Arguments

Frequently we'll want to give information to the function to work with, or on. In this case, we'll give the function one or more variables in which to store this information.

You can create a default value for an argument by simply assigning a value as you declare the function.

Return Values

Use the `return` keyword before a variable or value to output that value as the result of the function.

Modules

Modules are, essentially a collection of functions that are in a file that you can include in your program. For instance, if you have a lot of functions you use all the time, generally referred to as helper functions, you might save them in a file called **myHelperFunctions.py**. You can then use the import command to make those functions available in your code.

Our next workshop will go further into Modules.

Adding Functionality Through Modules

Built-In Modules

The core Python libraries provide a lot of functionality for basic programs. However, there is a lot more functionality available, written by other developers and researchers. But before we go off into the wonderful world of 3rd party modules, let's look at what comes with Python.

Open an IDLE instance and type the following:

```
>>> import sys
>>> sys.builtin_module_names
```

You should get output that looks something like this:

```
('_ast', '_bisect', '_codecs', '_codecs_cn', '_codecs_hk', '_codecs_iso2022',
'_codecs_jp', '_codecs_kr', '_codecs_tw', '_collections', '_csv', '_datetime',
'_functools', '_heapq', '_imp', '_io', '_json', '_locale', '_lsprof', '_md5',
'_multibytecodec', '_opcode', '_operator', '_pickle', '_random', '_sha1',
'_sha256', '_sha512', '_signal', '_sre', '_stat', '_string', '_struct', '_symtable',
'_thread', '_tracemalloc', '_warnings', '_weakref', '_winapi', 'array', 'atexit',
'audioop', 'binascii', 'builtins', 'cmath', 'errno', 'faulthandler', 'gc', 'itertools',
```

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

```
'marshal', 'math', 'mmap', 'msvcrt', 'nt', 'parser', 'sys', 'time', 'winreg',
'xxsubtype', 'zipimport', 'zlib')
```

This is a list of the modules that are built into Python and are available for use right now.

The `import` keyword tells the Python interpreter to load the code from another file and make the functions within available to your program.

To get more information about a module, you can use the `help()` function. This will list out all of the modules currently installed and registered with Python. Note: I had to truncate the listing for print.

```
>>> help('modules')

Please wait a moment while I gather a list of all available modules...


AutoComplete          _random               errno                 pyexpat
AutoCompleteWindow    _sha1                 faulthandler          pylab
AutoExpand            _sha256               filecmp               pyparsing
Bindings              _sha512               fileinput             pytz
CallTipWindow         _signal               fnmatch               queue
...
_osx_support          easy_install          pty                   xxsubtype
_overlapped           email                 py_compile            zipapp
_pickle               encodings             pyclbr                zipfile
_pydecimal            ensurepip             pydoc                 zipimport
_pyio                 enum                  pydoc_data            zlib

Enter any module name to get more help.  Or, type "modules spam" to
search
for modules whose name or summary contain the string "spam".
```

You can also use the `help()` function to get information on a specific module. First you will need to import the module. Again, the following listing was truncated for brevity.

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

```
>>> import math
>>> help(math)
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.

FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.


...

FILE
    (built-in)
```

Extended Modules

Finding Packages

In addition to the built-in modules that you get with every Python installation, there are countless extensions you can add, called packages. Fortunately, the good folks at Python have provided a method to learn about third party packages. Visit https://pypi.python.org/pypi for more information about 3rd party packages.

Installing Packages

Once you've found the package you want or need to install for your application, the easiest way to install it is using PIP. As of Python 2.7.9 and Python 3.4, the PIP binaries are included in the download. However, since the package is constantly evolving, you will likely need to upgrade it. If everything installed and configured correctly, you should be able to do this from the command line.

1. Open a terminal window
2. Depending on your OS:
    a. On Windows type:

```
python -m pip install -U pip
```

    b. On Linux or MacOS type:

```
pip install -U pip
```

Once that is done, you're ready to use PIP. One thing to keep in mind is you'll be running PIP from the terminal, not from within the Python shell.

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

For this demonstration, we'll be installing a package used for plotting mathematical formulas. This is a very popular package for visualizing data using Python. The actual use of this package is outside the scope of this workshop. For more information on using matplotlib, Google is your friend.

As a side note, and because I'm lazy, I'll be entering the commands for Windows. On Linux and MacOS you can simply truncate the `python -m` portion of the command.

To install a new package, type:

```
python -m pip install matplotlib
```

This will install the matplotlib library for your use.

Importing and Using Modules

Importing modules is easy. As you've seen earlier, you simply use the `import` keyword followed by the name of the module. This will load all of the functions of that module for your use. Now, in order to use one of the functions from the module, you will need to enter the module name followed by the function.

```
>>> import math
>>> math.sqrt(9)
3.0
```

Some packages, however are very large and you may not want to import the entire thing. If you know the specific function you will need in your program, you can import only that part of the module.

This will import the sqrt function from the math module. If you import just the function, you will not need to prefix the function with the module name.

```
>>> from math import sqrt
>>> sqrt(9)
3.0
```

Lastly, you can provide an alias for the modules and functions you import. This becomes very handy when you import a module with a fairly long name. In this example we're just being lazy…

```
>>> import math as m
>>> m.sqrt(9)
3.0
>>> from math import sqrt as s
>>> s(9)
3.0
```

Classes

Now we get to the good stuff. Classes. A class is nothing more than the logical representation of a physical or abstract entity within your code. For instance, a robot. The robot class will create a framework that we use to describe a physical robot to the program. How you describe it is entirely up to you, but is represented in how you build the class. This representation is abstract, in much the same way the word "robot" represents the

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

abstraction of the concept of a robot. If we were standing in a room full of robots, and I said, “hand me the robot,” your response is likely going to be, “which robot?” This is because the term “robot” applies to every robot in the room. But, if I were to say, “hand me Nomad,” you would know which, specific robot I was talking about. Nomad is an instance of a robot.

This is how a class is used. You will start by defining what the class is by constructing the abstraction of the entity you want to represent, in this case, a robot. Then, when you want to describe a specific robot, you will create an instance of the class that applies to that, specific robot.

There is a lot to learn about classes but here are the key things you need to know.

- A class is made up of functions called methods. These are the same thing as functions outside a class, with a couple exceptions.
- A method will always require the `self` parameter. This parameter is a reference to the instance of the class.
- The `self` parameter is always the first parameter of a method.
- Every class must have a special method called __init__. The __init__ method is called when an instance is created, and it initializes that instance of the class. In this method you will perform whatever needs to happen in order for the class to function. Most often you will define attributes for the class.
- Attributes of a class are variables within the class that describe some feature. For instance, for the robot class we would want a name and some functional attributes, like direction and speed.
- Methods are functions within a class that perform work. For instance, you may have method in the robot class called drive_forward(). In this method you would add the code for the robot to drive forward.

Types of Methods

- **Mutator Methods –** These are methods that change values within the class. For instance, setters are a type if mutator method that sets the value of an attribute. You can have other mutator methods that perform calculations or other functions to change values.
- **Accessor methods –** These are methods that access attributes within a class.
- **Helper methods –** These include any number of methods that perform work within the class. For example, the obligatory __init__ method is a type of helper called a constructor. Helper methods can be anything that performs work within class, generally for other methods. An example would be a helper method that formats a string prior to output.

Creating a Class

You can create a class anywhere in your code. However, what makes classes so useful is that they encapsulate functionality that allows you to easily port it from one project to the next. For this reason it is generally better to create a class as its own module and import it into your code. That is what we’ll be doing here.

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

So, at this point, let's build our robot class and then use it.

1. Create a new Python file and save it as "robot_sample.py".
2. Enter the following code:

```
class Robot():
    """A simple robot class"""
    def __init__(self,name,desc):
        """initilaizes our robot"""
        self.name = name
        self.desc = desc

    def drive_forward(self, distance):
        """simulates driving forward"""
        print(self.name.title() + " is driving forward " + str(distance) + "
meters")

    def stop(self):
        """siulates stopping the robot"""
        print(self.name.title() + " has stopped")

my_robot = Robot("Nomad","Rover")

print("My robot is a " + my_robot.desc + " called " + my_robot.name)
```

3. Save the file.
4. In the Python Shell window type:

```
>>> from robot_sample import Robot
>>> my_robot = Robot("Nomad","Autonomous rover")
>>> print("My robot is a " + my_robot.desc + " called " +
my_robot.name)
My robot is a Autonomous rover called Nomad
```

5. Type:

```
>>> my_robot.drive_forward(10)
Nomad is driving forward 10 meters
```

6. Type:

```
>>> my_robot.stop()
Nomad has stopped
```

Explore

Like everything else, we haven't even scratched the surface of functions and classes. The key thing is to know what you're looking at when you start examining other peoples' code. At this point you should be comfortable using the Python tools.

It is imperative that you practice what we went over here today. Start working with lists, tuples, and dictionaries. Start writing your own classes and functions. Start looking at other peoples' libraries and

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.

modules. In time, you're going to want to start building your own modules of helper functions. These are the functions you find yourself creating over and over again in your code. Slice these out into a separate file so all you have to do is import it.

We're not done using Python by a long shot. In upcoming workshops, we're going to be accessing the GPIO pins on the Python to interact with the real world, communicating with external devices and sensors, and building logic that will make the Pi do something in response to an input.

References:

https://www.python.org/

https://www.python.org/about/gettingstarted/

https://www.python.org/downloads/

https://www.python.org/about/

https://docs.python.org/3/tutorial/index.html

https://docs.python.org/3/library/idle.html

https://en.wikipedia.org/wiki/IDLE

"Python Crach Course" Eric Matthews, No Starch Press, 2016

©2016, Jeff Cicolani. All rights reserved. Used by The Robot Group with limited license.