Introduction to Robotics with the Raspberry Pi

INTRODUCTION TO PYTHON

Jeff Cicolani
Saturday, January 7, 2017

This 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 introduction to Python will be no different. We are going to show you how to install the tools, use the editor, and write some simple programs. We are going to move quickly through program structure, key syntax and formatting issues, data types, variables, and right into control structures and some of the object-oriented aspects of Python. Don't worry if any of this sounded like techno-babble, you'll understand it before you leave tonight (assuming you leave at the end of the workshop).

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).

Downloading and Installing Python

First, a little something on versions. There are, essentially, two flavors of Python being used; Python 2 and Python 3. With Python 3, the creator of Python, Guido van Rossum, decided to clean up the Python code without putting a lot of emphasis on backwards compatibility. As such, a lot of the code for version 2 simply won't work in version 3. Python 3 is the current version and everything will eventually move over to it. In fact, at this point, most everything has. In terms of robotics, the big hold out was OpenCV, an open source library of computer vision functions. There are some others that haven't migrated fully yet, so you'll need to figure out what you want to do and if the packages you need have been ported over yet. We will be using Python 3 for the workshop.

If you are using an Ubuntu or Debian Linux system such as the Raspberry Pi, you're done. The Python tools are already installed and ready to go. I don't know about the other flavors of Linux, you'll have to check. But most Debian based distributions install Python as part of the basic image. So, you're good. Go grab a drink while the Windows and iOS people play catch-up.

For the rest of you, let's get started.

1. Navigate to https://www.python.org/about/gettingstarted/
2. Click on Downloads
3. If you are using Windows, Click on "Download Python 3.6.0". This is the latest version at the time of this writing.
4. If you're using another OS, select the OS from the list following "Looking for Python with a different OS?" This will take you to the appropriate download link.

5. If you want to use an older release of Python, for some bizarre reason, you'll find the appropriate links by scrolling down the page.
6. Once it's downloaded, run the installer and follow the directions on the screen.

That's it. Now go get those Linux guys and tell them they're going to miss the rest of the fun.

IDLE: The Native Python Tools

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):

```
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!
```

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.

The Python Shell is launched when you open the native IDLE IDE. Depending who you ask, IDLE either stands for Integrated Development Environment or Integrated Development and Learning Environment. I like the latter simply because it makes more sense to me. But, in essence, it's a windowed interface to the Python interpreter. But it offers a lot of features you won't get in a simple command line.

- Simple editing features like find, copy, paste, etc.
- Syntax highlighting
- Text editor with syntax highlighting
- Debugger with stepping and breakpoints
- Many, many more

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.

As you'll learn later in this workshop, formatting is very important in Python. Other languages such as C, Java, etc., white space is irrelevant to the compiler. We use spaces, tabs, new lines, and blank lines to make the code more readable for people. In Python, however, indentation denotes code blocks. IDLE manages all of this for you. It will automatically indent your code for you, reducing the likelihood of syntax errors due to improper indentation.

There are also several tools to help you with your code. For instance, as you type, IDLE will present you with a list of possible statements appropriate for where you are in a line. There are a couple ways to invoke completions. Many times, it will automatically pop open while you are typing. This generally happens while you are inside a function call and there are only a limited number of possibilities. You can force it open by hitting `ctrl-space` while typing.

Another variant to completions is calltips. Calltips will display the expected values for an accessible function and will open when you type "(" after the function name. It will display the function signature and the first line of the docstring. It will remain open until the cursor is moved outside the function or the closing ")" is typed.

Context highlighting is done through colors. As you type your code, some of the words will change color. The colors have meanings and are a quick, visual way of verifying you are on the right track. Some of these are output, errors, user output and Python keywords, built-in class and function names, names following class and def, strings, and comments.

Let's see some of this in action.

1. If not already open, open IDLE
2. Click File → New File
   This will open a new text editor window
3. Type `pr`
4. Press `ctrl-space`
   This will display the completion list with the work "print" highlighted
5. Type `(`
   This does a couple things. It will select the highlighted text. In this case "print". It also displays the calltip for the print function.
6. Type `"Hello World"`
7. Type `)`
   The calltip will close after you type the closing ")"
8. Press enter
9. Save this file as "hello_world.py"
10. Press F5
    In the Python Shell window, you should see something like this:

```
RESTART: D:/Projects/The Robot Group/Workshops/Raspberry Pi
Robot/hello_world.py
Hello World
```

Oh, by the way, you just wrote and ran your first Python program.

Writing and Running a Python Program

Hello World

If you're following along like you should be then you've just written and run your first Python program. If you're not following along ("you rebel scum"), don't worry, we'll do it again right now but with a little more programming. Let's add a simple variable call. We'll talk about variables in the very near future.

1. If it's not already, open hello_world.py
2. If you're one of those rebels we mentioned earlier,
   a. If not already open, open IDLE
   b. Click **File → New File**
   c. Save this file as "hello_world.py"

Now that we're all on the same page…

3. Make the program look like this:

```
message = "Hello World"
print(message)
```

4. Save the file
5. Press F5 or select **Run → Run Module** from the menu.

   You should see the same output as before:

```
RESTART: D:/Projects/The Robot Group/Workshops/Raspberry Pi
Robot/hello_world.py
Hello World
```

All we did here is move the text from the print function into a variable and then told Python to print the contents of that variable.

Basic Structure

Program Parts

As you've seen, there aren't a lot of required parts for a Python program. Most programing languages require you to create, at the very least, a main function of some sort. For the Arduino it would be the `void loop()`. In c++ it would be `void main()`. Python does not have that. It will jump right into executing whatever commands it finds as it steps through the file. However, that does not mean it's entirely linear. We will be discussing functions and classes later in the workshop, but just know the interpreter will scan through the file and build whatever functions and classes it finds before executing the other commands. This is one of the things that makes Python so easy to learn. It simply doesn't have quite as rigid of framework as you would find in most other languages.

Oh, and for you programming language purists, Python walks the line between a scripting language, where everything is executed through the interpreter, and a programming language. It can be compiled into executables like c and c++. In fact, as we start to build modules, this is exactly what happens. However, most of the time, we'll simply be running it through the interpreter.

Indentation

As we're working through the workshop our programs will start getting more complex. In particular, we're going to start working with code blocks. Code blocks are commands that are grouped together to execute together in a loop or as part of a condition. This kind of structure is critical to being able to write effective programs.

All programming languages have a syntax for formatting code blocks. C based languages, including Java, will use brackets {} to contain a code block. Python does not do this. What Python does use is indentation. 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.

As we progress through the workshop you'll see the importance of indentation. In the meantime, just know it's important.

Comments

Commenting code has gotten more and more important over time. This is an area where programmers are notoriously deficient most of the time. They use comments, but they are, more frequently than not, cryptic or make assumptions about knowledge of the program that may nor hold true for someone who is picking up the code later. This would be a non-issue if they were any better about other forms of documentation.

My lamentations aside, commenting is important, especially as you're learning. So, get into the habit of using comments. Use them to tell you what you're doing or to enter small notes regarding logic.

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)
```

In the preceding code, we added two comment lines to our **hello_world.py** program. I also added a blank line to help make the code a little easier to read. If you were to save and run this program, you would get exactly the same output you did before.

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.

Running a Program

As we saw earlier, there are a couple ways to run a Python 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.

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.

```
> python hello_world.py
```

This would run our hello world program.

Variables and Data Types

For the next couple of sections, we will be using the Python shell and entering commands directly. A little later we will get back to writing program files. But, for now, everything we're doing can be done in the shell window.

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 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.

1. Using double quotes:

```
>>>print("This is text")
This is text
```

2. Using single quaotes

```
>>>print('This is text')
This is text
```

3. Single quotes inside double quotes:

```
>>>print("'This is text'")
```

```
'This is text'
```

4. Double quotes inside single quotes

```
>>>print('"This is text"')
"This is text"
```

You can add string variables together like this:

```
>>>first_name = "Jeff"
>>>last_name = "Cicolani"
>>>full_name = first_name + " " + last_name
>>>print(full_name)
Jeff Cicolani
```

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.

1. Integers:

```
>>>2+3
5
>>>3-2
1
>>>2*3
6
>>>3/2
1.5
>>>3**2
9
```

2. Floats:

```
>>>0.2+0.3
0.5
>>>0.4-0.2
0.2
>>>0.2*0.3
0.06
>>>2*0.3
0.6
```

There is one catch with Python floats, however. The interpreter will sometimes produce a seemingly arbitrary number of decimal places. This has to do with how the math is done within the interpreter.

```
>>>0.2+0.1
0.30000000000000004
```

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 [].

```
>>> robots = ["Nomad","Ponginator","Alfred"]
>>> print(robots)
['nomad', 'Ponginator', 'Alfred']
```

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.

```
>>> print(robots[0])
Nomad
```

There are a number of methods that are included with lists automatically. For example you can force the first letter of a name to be capitalized:

```
>>> print(robots[0].title())
Nomad
```

Working with Lists

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

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.

```
>>> colors = ("red","yellow","blue")
>>> print(colors)
('red', 'yellow', 'blue')
```

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

```
>>> Nomad = {"type":"rover","color":"black","processor":"Jetson TX1"}
>>> print(Nomad['type'])
Rover
```

You work with dictionaries in much the same way as you would an array. Except, rather than providing an index number, you provide the key to access an element.

Working with Dictionaries

Control Structures

In this section, we're going to explore how to add structure to your code. Rather than just stepping through a program and executing each line of code as it is encountered, you will probably want to have some more control. These control structures allow you to only execute code when a specific condition exists and to perform blocks of code multiple times.

For the most part it's going to be easier to walk you through these concepts rather than trying to describe them.

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.

```
>>> for robot in robots:
        if robot=="Nomad":
                print("This is Nomad")
        else:
                print(robot + " is not Nomad")


This is Nomad
Ponginator is not Nomad
Alfred is not Nomad
```

Again, note the indentation as you type. IDLE will indent another level after each line that ends in a colon, which should be every line that denotes a new block such as loop statements and conditionals.

It's important to also note how we test for equality. A single equals sign means assignment. Double equals signs means compare for equality.

```
>>> myRobot = "Nomad"
>>> myRobot == "Ponginator"
False
>>> myRobot == "Nomad"
True
```

Here is a list of comparators:

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

You can also use “and” and “or” to test for multiple conditions.

```
>>> for robot in robots:
        if robot == "Ponginator" or robot == "Alfred":
                print("These aren't the droids I'm looking for.")


These aren't the droids I'm looking for.
These aren't the droids I'm looking for.
```

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. As you enter this into the Python shell, pay attention to what it does with indentation. After you’ve entered the print command and pressed enter, you’ll need to press enter again so the shell knows you’re done.

```
>>> for robot in robots:
        print(robot)


nomad
Ponginator
Alfred
```

I tend to use plural for of words for my list names. This allows me to use the singular form of the name to reference items within the list for a loop.

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.

```
>>> count = 1
>>> while count < 5:
        print(count)
        count = count+1


1
2
3
4
```

In the previous code, had we forgotten to increment the count variable, it would have resulted in an open loop. Because count is equal to 1, and we never increment it, the value of count will always be less than 5. The code would never stop executing and we would need to press `ctrl+c` to end it.

Functions

Functions are predefined blocks of code that we can call from within the program to perform a task. We've been using the print() function throughout this workshop. There are many, many predefined functions in Python and many more that can be added using modules. For more on the functions available, check out the Python Standard Library: https://docs.python.org/3/library/index.html

But, there will many, many times you will want to create your own functions. Anytime you find yourself repeating the same set of operations throughout your code, you have a likely candidate for a function.

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.

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


>>> hello_world()
Hello World
```

In this code, we created a simple function that simply prints the message "Hello World". Now, whenever we want to print that message, we simply call that function.

```
>>> 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.

```
>>> def hello_user(first_name, last_name):
        print("Hello " + first_name + " " + last_name + "!")


>>> hello_user("Jeff","Cicolani")
Hello Jeff Cicolani!
```

We created a new function called hello_user and told it to expect to receive two pieces of information; the users first name and last name. The function simply prints the greeting.

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

```
>>> def favorite_thing(favorite = "robotics"):
        print("My favorite thing in the world is "+ favorite)


>>> favorite_thing("pie")
My favorite thing in the world is pie
>>> favorite_thing()
My favorite thing in the world is robotics
```

Note, the second time we called the function, we did not include a value. So, the function simply used the default value we assigned when we created the function.

Return Values

Sometime we don't just want the function to do something on its own. Sometimes we want it to give a value back to us.

```
>>> def how_many(list_of_things):
        count = len(list_of_things)
        return count

>>> how_many(robots)
3
```

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.

In Closing

Unfortunately, we don't have nearly enough time to go over everything in this workshop. In the next workshop we'll start to explore modules in more depth. We'll talk about how to import all or part of a module into your code. We'll use pip to install a couple common modules. We'll also talk about classes. These are more advanced topics but as easy to use as anything else we've covered today.

Explore

Admittedly, this was a lot of information. We certainly don't expect you to retain much, if any, of this. The point was to dip your toe in the water… then push you in. What we went over, here, is barely the beginning of your programming experiences when it comes to robotics. What we were hoping to accomplish with all of this was to get you comfortable with the interface, comfortable with working with code, and ready to branch out on your own.

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 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