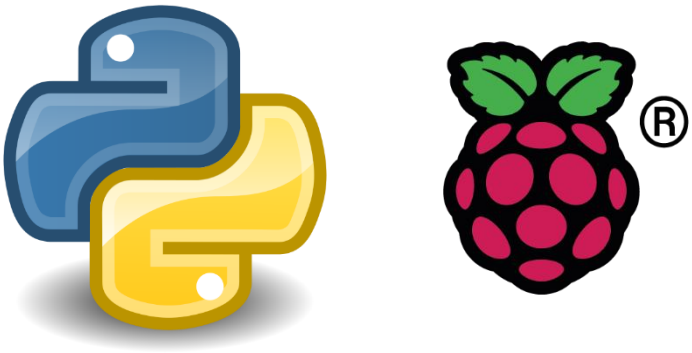

OpenCV & Raspberry Pi: Welcome to Computer Vision

A Fast Track Handbook

By Leong Tze Heng

Introduction

This handbook is specially designed to those who are interested in bringing automation in machine into reality through python programming. The handbook is made in such a way that users can learn about the most fundamental python programming syntax while familiarising to execute codes to run the automation in machine. In the OpenCV & Raspberry Pi: Welcome to Computer Vision Workshop, we would like to bring you the application of machine learning in Raspberry Pi by using Python as language of communication between the user and the machine.

The workshop aims to bring more enthusiastic learner or programmer to join the fascinating journey of computer vision. Throughout this workshop, it is hoped to expose beginners or experts of different field to computer vision from the eye of Pi. The focus of this workshop is the use of OpenCV library to be incorporated in Raspberry Pi! At the end of the workshop, the user can build a full understanding on image recognition using a Raspberry Pi and is able to build an image recognition model from scratch.

The handbook is divided into 3 sections, where the first section introduces about python programming for users to communicate with the Raspberry Pi. Second section will focus on the library – OpenCV as the library of programming functions for real-time computer vision. Last section focuses on the application of all knowledge to build a model for Raspberry Pi.

With that, I hope this handbook will be the guideline for those who wish to start their journey in Python, or in robotic fields or anyone has begun their journey but looking for more possibilities in this field.

Without further ado, get ready and happy coding!

Cheers,

Leong Tze Heng

Python Programming Language: The Basics

This section mainly focuses on describing Python as a general programming language for all and how it works as a programming language. At the end of this section, user should be able to understand what Python is, knowing Python syntax, understanding a Python script, writing simple Python commands and script, and running Python commands and script.

1.1 Python

Python is an **interpreted**, **multi-paradigm**, **high-level** programming language with **dynamic semantics** developed by Guido van Rossum. It is often used for building websites and software, automate tasks, and conduct data analysis, including machine learning purposes.

- **Interpreted vs Compiled programming language:**
  Case study example:
  Let say you have a ramen recipe that you want to cook tonight, but it's written in Japanese and there are two ways for you, a non-Japanese speaker to decipher and follow its instructions.

  The first method is if someone had already translated it into English for you. So, you and anyone who can speak English could read the English version of the recipe and make the ramen. Think of this translated recipe as the ***compiled*** version.

  The second method is if you have a friend or translator who knows Japanese. When you want to make the ramen, your friend or translator sits next to you and translates the recipe into English as you are cooking, line by line. In this case, your friend or translator is the ***interpreter*** for the ***interpreted*** version of the recipe.

  Compiled languages are converted directly into machine code that processor can execute. Hence, compiled languages need a "build" step - they need to be manually compiled first. Your program need to be rebuilt every time you make a change to the code. By using the case study example, the entire translation of the recipe is written before it is passed to you, hence, if there are any changes made to the recipe, the entire recipe would need to be translated again and resent to you.
  Example: C++, C, Java, Rust, Go

  For **interpreted languages**, **interpreters** run through a program line by line and execute each command. If the original author of the recipe decides to change his ingredients, he could just erase and add the new ingredients. Your

translator friend can still convey that change to you as it happens. Since, interpreted languages translate the program line by line at run time. Hence, the overall execution time is slower compared to compiled languages. Example: PHP, Ruby, Javascript, Perl

- **Multi-paradigm programming language:**
  Python can be written in procedural, object-oriented, functional or imperative manner. Most of the languages we use nowadays are multi-paradigm, such as Java, C++, JavaScript, Ruby, etc.

- **High level programming language:**
  Designed to be easier to understand and used by humans. Higher level of abstraction, which simplifies the programming process by defining complex data structures and operations using simpler and more intuitive syntax. Higher portability, which means the programs written in these languages can be run on different OS and hardware platforms with minimal modifications. Higher readability, which means designed to be easily readable and understandable by humans by using human-friendly syntax, meaningful variable and function names, and well structured code blocks. High productivity, which means aim to improve the productivity of developers by providing built-in libraries, frameworks, and tools that simplify common programming tasks contributed by a large community of developers.

- **Dynamic semantics:**
  Meaning that Python's variables are **dynamic objects.** Assign variable that make more sense to human than computer. Eg:
  C++:
  int x[3] = {1, 2, 3};

  Python:
  x = [1, 2, "three", "four"];

  ***Python is not type safety.**

1.2 Python Basics: The Beginning

This subsection aims to develop a very fundamental understanding of what python is and what python can do in simple executions. At the end of this subsection, user should know how to execute a simple code using Python and getting familiar with common syntaxes and concepts in Python.

1.2.1 Printing Output

The print() function is used to display output in Python. It takes one or more arguments and prints them to the console. You can pass strings, variables, or even mathematical expressions as arguments.

Example:

```
print("Hello, Python!")
```

1.2.2 Variables and Data Types

In Python, variables are used to store values. Unlike other programming languages, you don't need to explicitly declare the type of a variable in Python. The type of a variable is determined dynamically based on the value assigned to it. Here are some commonly used data types in Python:

- Integers: Whole numbers without decimals, e.g., 42, -10.
- Floats: Numbers with decimal points, e.g., 3.14, -0.5.
- Strings: Ordered sequences of characters, enclosed in single (') or double ("") quotes, e.g., "Hello, Python!".
- Booleans: Represents either True or False, used in logical operations.

To assign a value to a variable, use the assignment operator =. Here's an example:

```
age = 25

name = "John Doe"

is_student = True
```

1.2.3 Basic Arithmetic Operations

Python supports various arithmetic operations for numerical data types. Let's explore some common ones:

- Addition: +

- Subtraction: -
- Multiplication: *
- Division: /
- Modulo (Remainder): %
- Exponentiation: **

Example:

```
a = 10
b = 5


print(a + b)  # Output: 15
print(a - b)  # Output: 5
print(a * b)  # Output: 50
print(a / b)  # Output: 2.0
print(a % b)  # Output: 0
print(a ** b)  # Output: 100000
```

2.4 String Manipulation

Python provides powerful tools for working with strings. Let's explore some common string operations:

- Concatenation: Combining two or more strings using the + operator.
- Length: Getting the length of a string using the len() function.
- Indexing: Accessing individual characters in a string using square brackets [] and indices.
- Slicing: Extracting a portion of a string using the syntax [start:end].
- String methods: Python provides several built-in string methods for manipulating strings, such as lower(), upper(), replace(), split(), and more.

Example:

```
greeting = "Hello"
name = "John"
```

```
message = greeting + " " + name  # Output: "Hello John"
print(len(message))  # Output: 10
print(message[0])  # Output: "H"
print(message[6:9])  # Output: "Joh"
print(message.lower())  # Output: "hello john"
```

Checkpoint 1

Let's assess your understanding of the Python basics. Try to answer the following questions to test your knowledge:

1. What is the purpose of the print() function in Python?

a) To display output on the console

b) To perform mathematical operations

c) To store values in variables

d) To execute conditional statements

2. Which data type is used to represent decimal numbers in Python?

a) Integer

b) Float

c) String

d) Boolean

3. How would you concatenate two strings in Python?

a) Using the concat() function

b) Using the + operator

c) Using the join() method

d) Using the merge() function

4. Which of the following is an example of a conditional statement in Python?

a) for loop

b) while loop

c) if statement

d) try-except block

5. What is the output of the following code snippet?

```
numbers = [1, 2, 3, 4, 5]
print(numbers[2:4])
```

a) [2, 3]

b) [3, 4]

c) [2, 3, 4]

d) [1, 2, 3, 4]

ANSWER

a) To display output on the console

b) Float

b) Using the + operator

c) if statement

b) [3, 4]

1.3 Python Basics: The Intermediate

This subsection aims to expose the user with logics through computational thinking. User will understand the flow of the script and able to deduce the logics behind its execution while being able to generate their own flow to fit their usage purpose. At the end of this session, user should be able to know how to create a loop using conditional statements, creating functions and work with list.

1.3.1 Conditional Statements

Conditional statements allow us to execute different blocks of code based on specific conditions. The most used conditional statement in Python is the if statement. It follows the syntax:

```
if condition:
    # code to execute if condition is True
else:
    # code to execute if condition is False
```

Example:

```
age = 18


if age >= 18:
    print("You are an adult.")
else:
    print("You are not an adult.")
```

1.3.2 Loops

Loops are used to repeatedly execute a block of code. Python provides two types of loops: for loop and while loop.

1.3.2.1 **For** Loop

The for loop iterates over a sequence (such as a list, string, or range) and executes a block of code for each item in the sequence. It follows the syntax:

```
for item in sequence:
    # code to execute for each item
```

Example:

```
fruits = ["apple", "banana", "orange"]


for fruit in fruits:
    print(fruit)
```

1.3.2.2 **While** Loop

The **while** loop repeatedly executes a block of code if a certain condition is True. It follows the syntax:

```
while condition:
    # code to execute while condition is True
```

Example:

```
count = 1


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

1.3.3 Lists

Lists are ordered collections of items in Python. They can contain elements of different data types and are mutable, which means you can modify them after creation. Here's how you can create and manipulate lists:

1.3.3.1 Creating Lists

You can create a list by enclosing comma-separated values within square brackets [ ]. Here's an example:

```
fruits = ["apple", "banana", "orange"]
```

You can access individual elements of a list using indexing. Python uses zero-based indexing, which means the first element is at index 0. Here's an example:

```
print(fruits[0])  # Output: "apple"
print(fruits[1])  # Output: "banana"
```

1.3.3.2 Modifying Lists

Lists are mutable, which means you can modify their elements. Here are some common operations to modify lists:

- Adding elements: You can append an element to the end of a list using the **append()** method or insert an element at a specific position using the insert() method.

```
fruits.append("grape")      # Add "grape" at the end of the list
fruits.insert(1, "kiwi")    # Insert "kiwi" at index 1
print(fruits)               #
```

- Removing elements: You can remove an element from a list using the **remove()** method or remove an element at a specific index using the pop() method.

```
fruits.remove("banana")     # Remove "banana" from the list
fruits.pop(0)               # Remove the element at index 0
print(fruits)               #
```

- Updating elements: You can update the value of an element by assigning a new value to a specific index.

```
fruits[1] = "mango"         # Update the element at index 1 to "mango"
print(fruits)               #
```

1.3.4 Functions

Functions are blocks of reusable code that perform specific tasks. They help in organizing code and promoting reusability. Here's how you can define and use functions in Python:

1.3.4.1 Defining Functions

To define a function, use the def keyword followed by the function name, parentheses (), and a colon:. The function body is indented under the function definition.

```
def greet():
    print("Hello, Python!")
```

1.3.4.2 Calling Functions

To call a function, simply write the function name followed by parentheses ().

```
greet()  # Call the greet() function
```

Functions can also accept parameters, allowing you to pass values to the function. Here's an example:

```
def greet(name):
    print("Hello, " + name + "!")


greet("John")          # Call the greet() function with the argument "John"
```

Checkpoint 2

Let's assess your understanding of the intermediate Python concepts covered. Try to answer the following questions to test your knowledge:

1. Which type of loop is used when you want to repeatedly execute a block of code as long as a certain condition is True?

a) if statement

b) for loop

c) while loop

d) try-except block

2. How do you add an element to the end of a list in Python?

a) Using the insert() method

b) Using the append() method

c) Using the extend() method

d) Using the add() method

3. What is the purpose of defining a function in Python?

a) To store values

b) To execute conditional statements

c) To perform mathematical operations

d) To create reusable blocks of code

4. What will be the output of the following code snippet?

```
count = 0
while count < 5:
    print(count)
    count += 2
```

a) 0 1 2 3 4

b) 0 2 4

c) 0 1 2 3

d) 1 3 5

5. How do you access the length of a string in Python?

a) Using the length() method

b) Using the count() method

c) Using the size() method

d) Using the len() function

ANSWER

c) while loop

b) Using the append() method

d) To create reusable blocks of code

b) 0 2 4

d) Using the len() function

Challenge yourself!

1. Sum of Even Numbers

   Write a Python program that prompts the user to enter a positive integer. Then, calculate and display the sum of all even numbers from 1 to that entered number.

2. Sorting Numbers in a List

   Write a Python program that takes a list of numbers as input and sorts the numbers in ascending order using a sorting algorithm of your choice. You should implement the sorting algorithm from scratch, rather than using built-in functions like **sorted()** or **sort().**

3. Alphabet Counter

   Write a Python program that takes a string as input and counts the frequency of each character in the string. Then, display the characters along with their respective frequencies in alphabetical order.

4. Sieve of Erathosthenes

   Sieve if Eratosthenes is an ancient algorithm that use iterative method to find all prime numbers up to a given number. The procedures below are the operations to find prime number up to 30.

   1. Generate a list of integers from 2 to 30.

   2. Start from 2, remove the number that are divisible by 2.

   3. Next is 3, remove the number that divisible by 3.

   4. Number of 4 is removed at step 2, so remove the number that divisible by 5.

   5. The operation is stopped here because the next number close to 5 is 7, and 7 x 7 is 49 >30(input).

   Write a Python code to execute Sieve of Eratosthenes to find all the prime numbers in 100.

5. Binary Search

   Write a Python code to implement a binary search algorithm. Given a sorted list of numbers and a target number, your task is to find the index of the target number in the list, or return -1 if the target number is not present.