Dedicated To

My Family

Who supported me thick and thinPreface

First, Congratulations on investing your hard-earned money on this book. Who should read this book past this page?? A Techie by heart, A Hobbyist by nature & A Hacker by chance. Someone looking forward to having a rocking time with this magical box, labeled Raspberry Pi.

The Author was as excited as you are right now to begin his hacking spree, but it was when I faced numerous challenges (both on Hardware and Software level), that I decided to capture my daily learning’s in the form of simple notes on notepad. And then, it all started, with my collection of simple notes to capture new learning’s, which has now taken the shape of a book.

The basic idea behind publishing this book, is to cut short the plethora of information available on various websites on how-to build System Image for Raspberry-Pi, and to help readers quickly jump to the practical stuff that matters, without wasting time and energy on.

Everything said and done, I would personally love to hear from you, your suggestions to make this book even better.

Regards,

Sudhanshu Gupta

Founder, CEO

[www.softwaresunleashed.com](http://www.softwaresunleashed.com)

E-mail: softwares.unleashed@gmail.com

# About the Author

Sudhanshu Gupta (Founder – Softwares Unleashed), is a Bachelors in Electronics & Tele-Communications & M.S. in Softwares Systems and has been associated with Telecom domain and Embedded Software development since 2004.

He had worked with Major Industry giant(s), LG, Infineon, Intel to name a few. Sudhanshu during his stint with the corporate world, has contributed to numerous success stories of Big OEMs (LG, Samsung, Nokia) …cutting short the list.

He is now on a fast track to take his passion forward, i.e. Application Development and Sharing his Technical Knowledge for the benefit of others.

# Before we start

Hope my work would be helpful to most of you reading this book, and enjoy working on the demo projects as much as I loved it creating them.

So without any further ado, let’s get started…

# Table of Contents

About the Author 3

Before we start 3

Table of Contents 4

About Artificial Intelligence 6

Support Repo 7

Python 2 vs Python 3 8

Necessary Packages Installation 9

1. Install the Python development environment on your system 9

Ubuntu 9

Mac OS 9

Windows 9

Raspberry Pi 9

2. Create a virtual environment (recommended) 10

Ubuntu / Mac OS 10

Windows 10

3. Install the TensorFlow pip package 10

Virtual Env Install 10

System Install 11

Google Colaboratory – Colab 12

The Basics 12

Using Colab 12

Additional Questions 15

AI & Raspberry Pi 16

Trouble Shooting Guide 18

ModuleNotFoundError: No module named 'tensorflow' 18

Reference(s) 24

Further Reading 25

Legends 26

Revision History 27

# About Artificial Intelligence

TODO: Add some intro to Artificial Intelligence.

.

#### Overview

TODO : Add Some more overview of AI.

# Support Repo

This guide comes equipped with support material to speed up and enhance your learning experience, by collating all the necessary tools in a single repo. Just in case we missed it, you need to have an account created on [www.github.com](http://www.github.com) to access this support repo.

Git Repo Link: <https://github.com/softwaresunleashed/ai.git>

Create an empty folder and type in following command on shell prompt (and wait for few mins):

$ git clone --recursive https://github.com/softwaresunleashed/ai.git

Note: GIT is a pre-requisite to this step, hence, if GIT isn’t installed yet on your environment, execute the following command to install GIT on your system

$ sudo apt-get install git

# Python 2 vs Python 3

## What is Python 2?

Python 2 made code development process easier than earlier versions. It implemented technical details of Python Enhancement Proposal (PEP). Python 2.7 (last version in 2.x ) is no longer under development and in 2020 will be discontinued.

## What is Python 3?

On December 2008, Python released version 3.0. This version was mainly released to fix problems which exist in Python 2. The nature of these change is such that Python 3 was incompatible with Python 2. It is **backward incompatible** Some features of Python 3 have been backported to Python 2.x versions to make the migration process easy in Python 3.

As a result, for any organization who was using Python 2.x version, migrating their project to 3.x needed lots of changes. These changes not only relate to projects and applications but also all the libraries that form part of the Python ecosystem.

## Why should you learn Python 2?

Although, Python 2 is an old open source version here are where you still need to learn Python 2:

* To become a DevOps engineer and you need to work with configurations management tools like puppet or ansible. Here, you need to work with both of these versions.
* If your company's code written in Python 2, you will require to learn to work with that
* If your development team is working on a project that depends on specific third-party libraries or software which you are not able to port to Python 3, then Python 2 is the only option available for you.

## Why should you use Python 3?

Here, are prime reasons for using Python 3.x versions:

* Python 3 supports modern techniques like AI, machine learning, and data science
* Python 3 is supported by a large Python developer's community. Getting support is easy.
* Its easier to learn Python language compared to earlier versions.
* Offers Powerful toolkit and libraries
* Mixable with other languages

## History of Python 2

* Python 2.0 - October 16, 2000
* Python 2.1 - April 17, 2001
* Python 2.2 - December 21, 2001
* Python 2.3 - July 29, 2003
* Python 2.4 - November 30, 2004
* Python 2.5 - September 19, 2006
* Python 2.6 - October 1, 2008
* Python 2.7-July 3, 2010

## History of Python 3

* Python 3.0 - December 3, 2008
* Python 3.1 - June 27, 2009
* Python 3.2 - February 20, 2011
* Python 3.3 - September 29, 2012
* Python 3.4-March 16, 2014
* Python 3.5 - September 13, 2015
* Python 3.6- October 2016
* Python 3.7- June 2018.

## Key Differences Between Python 2 and Python 3

|  |  |  |
| --- | --- | --- |
| Basis of comparison | Python 3 | Python 2 |
| Release Date | 2008 | 2000 |
| Function print | print ("hello") | print "hello" |
| Division of Integers | Whenever two integers are divided, you get a float value | When two integers are divided, you always provide integer value. |
| Unicode | In Python 3, default storing of strings is Unicode. | To store Unicode string value, you require to define them with "u". |
| Syntax | The syntax is simpler and easily understandable. | The syntax of Python 2 was comparatively difficult to understand. |
| Rules of ordering Comparisons | In this version, Rules of ordering comparisons have been simplified. | Rules of ordering comparison are very complex. |
| Iteration | The new Range() function introduced to perform iterations. | In Python 2, the xrange() is used for iterations. |
| Exceptions | It should be enclosed in parenthesis. | It should be enclosed in notations. |
| Leak of variables | The value of variables never changes. | The value of the global variable will change while using it inside for-loop. |
| Backward compatibility | Not difficult to port python 2 to python 3 but it is never reliable. | Python version 3 is not backwardly compatible with Python 2. |
| Library | Many recent developers are creating libraries which you can only use with Python 3. | Many older libraries created for Python 2 is not forward-compatible. |

## Python 2 vs. Python 3 Example Code

**Python 3**

def main():

print("Hello World!")

if \_\_name\_\_== "\_\_main\_\_":

main()

**Python 2**

def main():

print "Hello World!"

if \_\_name\_\_== "\_\_main\_\_":

main()

## Which Python Version to Use?

When it comes to Python version 2 vs. 3 today, Python 3 is the outright winner. That's because Python 2 won't be available after 2020. Mass Python 3 adoption is the clear direction of the future.

After considering declining support for Python 2 programming language and added benefits from upgrades to Python 3, it is always advisable for a new developer to select Python version 3. However, if a job demands Python 2 capabilities, that would be an only compelling reason to use this version.

# Necessary Packages Installation

## 1. Install the Python development environment on your system

$ python3 --version  
$ pip3 --version  
$ virtualenv --version

If these packages are already installed, skip to the next step. Otherwise, install [Python](https://www.python.org/), the [pip package manager](https://pip.pypa.io/en/stable/installing/), and [Virtualenv](https://virtualenv.pypa.io/en/stable/):

### Ubuntu

$ sudo apt update

$ sudo apt install python3-dev python3-pip

$ sudo pip3 install -U virtualenv # system-wide install

### Mac OS

$ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

$ export PATH="/usr/local/bin:/usr/local/sbin:$PATH"  
$ brew update  
$ brew install python   # Python 3  
$ sudo pip3 install -U virtualenv   # system-wide install

### Windows

Install the Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017, and 2019. Starting with the TensorFlow 2.1.0 version, the msvcp140\_1.dll file is required from this package (which may not be provided from older redistributable packages). The redistributable comes with Visual Studio 2019 but can be installed separately:

1. Go to the [Microsoft Visual C++ downloads](https://support.microsoft.com/en-us/help/2977003/the-latest-supported-visual-c-downloads/),
2. Scroll down the page to the Visual Studio 2015, 2017 and 2019 section.
3. Download and install the Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019 for your platform.

Make sure [long paths are enabled](https://superuser.com/questions/1119883/windows-10-enable-ntfs-long-paths-policy-option-missing) on Windows.

Install the 64-bit [Python 3 release for Windows](https://www.python.org/downloads/windows/) (select pip as an optional feature).

C:\> pip3 install -U pip virtualenv

### Raspberry Pi

$ sudo apt update

$ sudo apt install python3-dev python3-pip

$ sudo apt install libatlas-base-dev # required for numpy

$ sudo pip3 install -U virtualenv # system-wide install

## 2. Create a virtual environment (recommended)

### Ubuntu / Mac OS

Create a new virtual environment by choosing a Python interpreter and making a ./venv directory to hold it:

$ virtualenv --system-site-packages -p python3 ./venv

Activate the virtual environment using a shell-specific command:

$ source ./venv/bin/activate   # sh, bash, ksh, or zsh

When virtualenv is active, your shell prompt is prefixed with (venv).

Install packages within a virtual environment without affecting the host system setup.

Start by upgrading pip:

(venv) $ pip install --upgrade pip  
(venv) $ pip list   # show packages installed within the virtual environment

And to exit virtualenv later:

(venv) $ deactivate   # don't exit until you're done using TensorFlow

### Windows

Create a new virtual environment by choosing a Python interpreter and making a .\venv directory to hold it:

C:\> virtualenv --system-site-packages -p python3 ./venv

Activate the virtual environment:

(venv) C:\> .\venv\Scripts\activate

Install packages within a virtual environment without affecting the host system setup. Start by upgrading pip:

(venv) C:\> pip install --upgrade pip  
  
(venv) C:\> pip list   # show packages installed within the virtual environment

And to exit virtualenv later:

(venv) C:\> deactivate   # don't exit until you're done using TensorFlow

## 3. Install the TensorFlow pip package

### Virtual Env Install

(venv) $ pip install --upgrade tensorflow

Verify the install:

(venv) $ python -c "import tensorflow as tf;print(tf.reduce\_sum(tf.random.normal([1000, 1000])))"

### System Install

$ pip3 install --user --upgrade tensorflow  # install in $HOME

Verify the install:

$ python3 -c "import tensorflow as tf; print(tf.reduce\_sum(tf.random.normal([1000, 1000])))"

# Google Colaboratory – Colab

[ Ref : <https://research.google.com/colaboratory/faq.html> ]

## The Basics

**What is Colaboratory?**

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs.

**Is it really free to use?**

Yes. Colab is free to use.

**Seems too good to be true. What are the limitations?**

Colab resources are not guaranteed and not unlimited, and the usage limits sometimes fluctuate. This is necessary for Colab to be able to provide resources for free.

**What is the difference between Jupyter and Colab?**

[Jupyter](https://jupyter.org/) is the open source project on which Colab is based. Colab allows you to use and share Jupyter notebooks with others without having to download, install, or run anything.

## Using Colab

**Where are my notebooks stored, and can I share them?**

Colab notebooks are stored in [Google Drive](https://drive.google.com/), or can be loaded from [GitHub](https://github.com/). Colab notebooks can be shared just as you would with Google Docs or Sheets. Simply click the Share button at the top right of any Colab notebook

**If I share my notebook, what will be shared?**

If you choose to share a notebook, the full contents of your notebook (text, code, output, and comments) will be shared. You can omit code cell output from being saved or shared by using **Edit > Notebook settings > Omit code cell output when saving this notebook.** The virtual machine you’re using, including any custom files and libraries that you’ve setup, will not be shared. So it’s a good idea to include cells which install and load any custom [libraries](https://colab.research.google.com/notebooks/snippets/importing_libraries.ipynb) or [files](https://colab.research.google.com/notebooks/io.ipynb) that your notebook needs.

**Can I import an existing Jupyter/IPython notebook into Colab?**

Yes. Choose "Upload notebook" from the File menu.

**How can I search Colab notebooks?**

You can search Colab notebooks using [Google Drive](https://drive.google.com/). Clicking on the Colab logo at the top left of the notebook view will show all notebooks in Drive. You can also search for notebooks that you have opened recently using **File > Open Recent**.

**Where is my code executed? What happens to my execution state if I close the browser window?**

Code is executed in a virtual machine private to your account. Virtual machines are deleted when idle for a while, and have a maximum lifetime enforced by the Colab service.

**How can I get my data out?**

You can download any Colab notebook that you’ve created from Google Drive, or from within Colab’s File menu. All Colab notebooks are stored in the open source Jupyter notebook format ( .ipynb).

**How can I reset the virtual machine(s) my code runs on, and why is this sometimes unavailable?**

Selecting **Runtime > Factory reset runtime** to return all managed virtual machines assigned to you to their original state. This can be helpful in cases where a virtual machine has become unhealthy e.g. due to accidental overwrite of system files, or installation of incompatible software. Colab limits how often this can be done to prevent undue resource consumption. If an attempt fails, please try again later.

**Why does drive.mount() sometimes fail saying "timed out", and why do I/O operations in drive.mount()-mounted folders sometimes fail?**

Google Drive operations can time out when the number of files or subfolders in a folder grows too large. If thousands of items are directly contained in the top-level "My Drive" folder then mounting the drive will likely time out. Repeated attempts may eventually succeed as failed attempts cache partial state locally before timing out. If you encounter this problem, try moving files and folders directly contained in "My Drive" into sub-folders. A similar problem can occur when reading from other folders after a successful drive.mount(). Accessing items in any folder containing many items can cause errors like OSError: [Errno 5] Input/output error (python 3) or IOError: [Errno 5] Input/output error (python 2).

Again, you can fix this problem by moving directly contained items into sub-folders.  
Note that "deleting" files or subfolders by moving them to the Trash may not be enough; if that doesn't seem to help, make sure to also [Empty your Trash](https://support.google.com/drive/answer/2375102#permanent).

**Why aren’t resources guaranteed in Colab?**

In order to be able to offer computational resources for free, Colab needs to maintain the flexibility to adjust usage limits and hardware availability on the fly. Resources available in Colab vary over time to accommodate fluctuations in demand, as well as to accommodate overall growth and other factors.

Some users want to be able to do more in Colab than the resource limits allow. We have heard from many users who want faster GPUs, longer running notebooks and more memory, as well as usage limits that are higher and don’t fluctuate as much. Introducing [Colab Pro](http://colab.research.google.com/signup?utm_source=faq&utm_medium=link&utm_campaign=why_arent_resources_guaranteed) is the first step we are taking towards serving users who want to do more in Colab. Our long term goal is to continue providing a free version of Colab, while also growing in a sustainable fashion to meet the needs of our users. If you are interested in doing more in Colab than the resource limits of the free version of Colab allow, please try out Colab Pro and let us know what you think.

**What are the usage limits of Colab?**

Colab is able to provide free resources in part by having dynamic usage limits that sometimes fluctuate, and by not providing guaranteed or unlimited resources. This means that overall usage limits as well as idle timeout periods, maximum VM lifetime, GPU types available, and other factors vary over time. Colab does not publish these limits, in part because they can (and sometimes do) vary quickly.

GPUs and TPUs are sometimes prioritized for users who use Colab interactively rather than for long-running computations, or for users who have recently used less resources in Colab. As a result, users who use Colab for long-running computations, or users who have recently used more resources in Colab, are more likely to run into usage limits and have their access to GPUs and TPUs temporarily restricted. Users with high computational needs may be interested in using Colab’s UI with a [local runtime](https://research.google.com/colaboratory/local-runtimes.html) running on their own hardware. Users interested in having higher and more stable usage limits may be interested in [Colab Pro](http://colab.research.google.com/signup?utm_source=faq&utm_medium=link&utm_campaign=what_are_usage_limits).

**What types of GPUs are available in Colab?**

The types of GPUs that are available in Colab vary over time. This is necessary for Colab to be able to provide access to these resources for free. The GPUs available in Colab often include Nvidia K80s, T4s, P4s and P100s. There is no way to choose what type of GPU you can connect to in Colab at any given time. Users who are interested in more reliable access to Colab’s fastest GPUs may be interested in [Colab Pro](http://colab.research.google.com/signup?utm_source=faq&utm_medium=link&utm_campaign=what_types_of_gpus).

Note that using Colab for cryptocurrency mining is disallowed entirely, and may result in your account being restricted for use with Colab altogether.

**How long can notebooks run in Colab?**

Notebooks run by connecting to virtual machines that have maximum lifetimes that can be as much as 12 hours. Notebooks will also disconnect from VMs when left idle for too long. Maximum VM lifetime and idle timeout behavior may vary over time, or based on your usage. This is necessary for Colab to be able to offer computational resources for free. Users interested in longer VM lifetimes and more lenient idle timeout behaviors that don’t vary as much over time may be interested in [Colab Pro](http://colab.research.google.com/signup?utm_source=faq&utm_medium=link&utm_campaign=how_long_can_nbs_run).

**How much memory is available in Colab?**

The amount of memory available in Colab virtual machines varies over time (but is stable for the lifetime of the VM). (Adjusting memory over time allows us to continue to offer Colab for free.) You may sometimes be automatically assigned a VM with extra memory when Colab detects that you are likely to need it. Users interested in having more memory available to them in Colab, and more reliably, may be interested in [Colab Pro](http://colab.research.google.com/signup).

**How can I get the most out of Colab?**

Resources in Colab are prioritized for users who have recently used less resources, in order to prevent the monopolization of limited resources by a small number of users. To get the most out of Colab, consider closing your Colab tabs when you are done with your work, and avoid opting for a GPU when it is not needed for your work. This will make it less likely that you will run into usage limits in Colab. Users interested in going beyond the resource limits in the free version of Colab may be interested in [Colab Pro](http://colab.research.google.com/signup).

**I saw a message saying my GPU is not being utilized. What should I do?**

Colab offers optional accelerated compute environments, including GPU and TPU. Executing code in a GPU or TPU runtime does not automatically mean that the GPU or TPU is being utilized. To avoid hitting your GPU [usage limits](https://research.google.com/colaboratory/faq.html#usage-limits), we recommend switching to a standard runtime if you are not utilizing the GPU. Choose **Runtime > Change Runtime Type** and set **Hardware Accelerator** to **None**.

For examples of how to utilize GPU and TPU runtimes in Colab, see the [Tensorflow With GPU](https://colab.research.google.com/notebooks/gpu.ipynb) and [TPUs In Colab](https://colab.research.google.com/notebooks/tpu.ipynb) example notebooks.

## Additional Questions

**What browsers are supported?**

Colab works with most major browsers, and is most thoroughly tested with the latest versions of [Chrome](https://www.google.com/chrome/browser/desktop/index.html), [Firefox](https://www.mozilla.org/en-US/firefox/) and [Safari](https://www.apple.com/safari/).

**How is this related to colaboratory.jupyter.org?**

In 2014 we worked with the Jupyter development team to release an early version of the tool. Since then Colab has continued to evolve, guided by internal usage.

**What about other programming languages?**

Colab focuses on supporting Python and its ecosystem of third-party tools. We're aware that users are interested in support for other Jupyter kernels (eg R or Scala). We would like to support these, but don't yet have any ETA.

**I found a bug or have a question, who do I contact?**

Open any Colab notebook. Then go to the Help menu and select ”Send feedback...”.

**Why prompt to enable third-party cookies?**

Colab uses HTML iframes and service workers hosted on separate origins in order to display rich outputs securely. Browsers [require enabling third-party cookies to use the service workers within iframes](https://dev.chromium.org/Home/chromium-security/security-faq/service-worker-security-faq#TOC-Can-iframes-register-Service-Workers-). An alternative to enabling third-party cookies for all sites is to whitelist the following hostname in your browser settings: googleusercontent.com.

**When will Colab phase-out Python 2?**

The Python development team has declared that Python 2 will no longer be supported after [January 1st, 2020](https://www.python.org/dev/peps/pep-0373/#maintenance-releases). As of that date, Colab has stopped updating Python 2 runtimes, and will begin phasing out support for Python 2 in the following months.

For users, this means that for the time being, Python 2 notebooks will continue to work. However, at some point in the future Python 2 notebooks may begin to connect to a Python 3 runtime instead. To avoid surprises, we suggest migrating important notebooks to Python 3.

To change your notebook's runtime, choose **Runtime > Change Runtime Type** and select **Python 3**. For information on migrating your code from Python 2 to Python 3, see [Porting Python 2 Code to Python 3](https://docs.python.org/3/howto/pyporting.html) within Python's documentation.

**Where can I learn more about Colab Pro?**

There is an FAQ for Colab Pro on the [Colab Pro sign-up page](https://colab.research.google.com/signup).

# AI & Raspberry Pi

This section contains IoT based fun (and serious) projects. The projects described below touch upon all the major aspects used by hobbyist, engineers, scientists, hackers in the past and till date. Knowledge of these Hardware & Software components

TODO : Add Projects related to AI & Raspberry Pi here.

# Trouble Shooting Guide

Following section provides real life problems encountered and Tips & Tricks on how they were fixed. Priceless!!!

## ModuleNotFoundError: No module named 'tensorflow'

Primarily, tensorflow isn’t installed on your system. Check the “Necessary Packages Installation” for any missing modules.

# Reference(s)

Python2 vs Python3 : <https://www.guru99.com/python-2-vs-python-3.html>

# Further Reading

# Legends

AI – Artificial Intelligence

TPU – Tensor Processing Unit (TPU) is an AI accelerator application-specific integrated circuit (ASIC) developed by Google specifically for neural network machine learning.

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Remarks** | **Author** |
|  |  |  |  |
| 0.1 | 18 Mar 2020 | Initial Draft | S.G. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |