Deep Dive into the Microsoft Professional Program in Artificial Intelligence - Part 1

=====================

Chris Cameron, Spiria

# Background

Recently I found some downtime at Spiria and decided to do some skills upgrading. I have long had an interest in Artificial Intelligence stemming from University courses in AI, Discrete Search and Optimization, and Matrix Calculations, among others. I was happy to find that Microsoft had opened up an extensive internal training program to the general public through a partnership with [edX](https://www.edx.org), so upon finding it I began auditing the program.

The whole program is broken up into 10 courses, which are:

1. Introduction to Artificial Intelligence (AI)

2. Introduction to Python for Data Science

3. Essential Math for Machine Learning: Python Edition

4. Ethics and Law in Data and Analytics

5. Data Science Research Methods: Python Edition

6. Principles of Machine Learning: Python Edition

7. Deep Learning Explained

8. Reinforcement Learning Exlained

9. Develop Applied AI Solutions (three options below)

- Computer Vision and Image Analysis

- Speech Recognition Systems

- Natural Language Processing (NLP)

10. Final Project

To help solidify some of the knowledge I learned throughout the courses, I will be writing a series of articles that tackle everything from technical troubleshooting, my favourite parts, to ethics.

# Introduction to Artificial Intelligence (AI) - Course Setup

In this first entry I'm going to discuss the first course in the program, [Introduction to Artificial Intelligence (AI)](https://www.edx.org/course/introduction-artificial-intelligence-3). There is some foundational setup required if you wish to run parts of this course on your local machine, as opposed to entirely in Microsoft's cloud. Of course there's nothing wrong with doing everything in the cloud if that's your thing; however I appreciate having at least some understanding of the setup, installation, and configuration of the tools I'm using. Also, during the course, I frequently experienced issues accessing Azure's [Jupyter Notebooks](htttps://notebooks.azure.com) service, so it's always good to have a local version just in case.

# Course Setup Basics

The setup guide for the course mentions that optionally you can install what's known as the Anaconda Python distribution. If you've never heard of this, you may be surprised to learn that the Python language is actually distributed in a few interesting ways outside of just swapping out the runtime. You may be familiar with CPython which is the standard runtime, or Jython, the Java-based Python, or IronPython, the .NET-based Python.

However there are other distributions which "bundle up" Python with additional libraries, package managers, web applications, and more. Two popular distributions are Anaconda Python and ActivePython. If this is all Klingon to you, take a peak at [this article](https://www.infoworld.com/article/3267976/python/anaconda-cpython-pypy-and-more-know-your-python-distributions.html) which does a nice job of summing up the basic landscape of Python distributions.

In this course, we'll use Anaconda Python, which is described by Anaconda Inc. as:

> the world’s most popular data science distribution for Python and R. Anaconda, Inc. continues to lead open source projects like Jupyter and Pandas that form the foundation of modern data science.

Basically, Anaconda comprises CPython-based Python, Jupyter Notebooks, Pandas, and hundreds of other data science libraries. Its big distinction is that it doesn't just manage Python packages (i.e., with `pip`) as it also can manage external dependencies as well with the `conda` package manager. Additional features include the Anaconda desktop GUI which allows you to control all of these tools with a familiar graphical interface. I'm one of those coders who prefers the terminal so I'll be sharing with you how to do all this from the terminal.

# Getting Anaconda

Microsoft doesn't get specific in the course about how to do this except to send you to [Anaconda's website](https://anaconda.com). However in this era of software development I'd be remiss not to show you a very nice container-based way to get started here.

If you're not familiar with Docker I'd suggest you brush up there. A great course on this is [Dive into Docker](https://diveintodocker.com) by Nick Janetakis, and of course there are free tutorials all over the web on how to setup, install, and use Docker.

One more note before continuing. There are two official docker images available for Anaconda on DockerHub. One is the full Anaconda distribution with GUI and all. The second is called "MiniConda", which is much smaller and allows you to use `conda` directly to download packages as you need them. MiniConda is my preference, but if you've got the bandwidth and don't feel like playing around with package management then feel free to use the [Anaconda image](https://hub.docker.com/r/continuumio/anaconda3) instead of the [Miniconda image](https://hub.docker.com/r/continuumio/miniconda3)

To grab miniconda, open your terminal and type the following:

```bash

$ docker pull continuumio/miniconda3

```

This will run for a while pulling the necessary files locally.

# Installing and Running Jupyter

While I was taking the course I ended up having to correct the instructions listed in the [README.md](https://github.com/ContinuumIO/docker-images/tree/master/miniconda3) but they have not propagated to DockerHub yet. The key point is the `--ip` flag needs to be set to `--ip='0.0.0.0'`.

The command to run the container and start Jupyter Notebook server is:

```bash

$ docker run -i -t -p 8888:8888 continuumio/miniconda3 /bin/bash -c "/opt/conda/bin/conda install jupyter -y --quiet && mkdir /opt/notebooks && /opt/conda/bin/jupyter notebook --notebook-dir=/opt/notebooks --ip='0.0.0.0' --port=8888 --no-browser"

```

Let's break this command down:

- `docker run -i -t -p 8888:8888 continuumio/miniconda3` - Run the `miniconda3` container with an interactive terminal and map port 8888 to local port 8888

- `/bin/bash -c "..."` - Run `bash` and give it a set of commands (`-c`) specified in the string wrapped in double quotes `""`

- `/opt/conda/bin/conda install jupyter -y --quiet` - the first command provided runs `conda` the package manager. It installs Jupyter in `--quiet` mode (less terminal output) and automatically answers yes (`-y`) to any installation questions

- `&& /opt/conda/bin/jupyter notebook --notebook-dir=/opt/notebooks --ip='0.0.0.0' --port=8888 --no-browser` - the second command runs the Jupyter server telling it to store notebooks in `/opt/notebooks`, expose the web interface on localhost (`0.0.0.0`) on port 8888, and don't launch a browser (`--no-browser`)

There's a lot going on there. And since you want this container to stay up for a while so you can play/learn, I recommend actually just trying to run the container with `bash` and then enter the commands manually, i.e., use `docker run -i -t -p 8888:8888 continuumio/miniconda3 /bin/bash` and then once you have the interactive terminal, run the Jupyter installation yourself (maybe even get rid of `--quiet` and `-y` so you can examine the output) and launch Jupyter after that.

\_Note: If you get an error about not allowing to run as root, you can add `--allow-root` to the end of the second command, right after `--no-browser`.\_

# Verify Everything is OK

To verify that everything has worked well, open another terminal and type the following commands:

```bash

$ docker exec -it FUNKY\_NAME\_OF\_CONTAINER bash`

$ conda update

$ python --version

$ conda list

```

The first command \_executes\_ `bash` inside of an interactive terminal to the `FUNKY\_NAME\_OF\_CONTAINER` container (You can find out the name of your container by typing `docker container ls`, find the entry for `continuumio/miniconda3` and copying out the `NAME` field; for example, mine was `romantic\_gates`).

\_Note: It's important that you understand that `docker run` starts a new container, whereas `docker exec` executes a command in an \*\*already running\*\* container\_

Once you're attached to the container, you can run the rest of the commands and examine their output which should be some pretty benign output, without warnings.

As a final verification, head to [http://127.0.0.1:8888](http://127.0.0.1:8888) and make sure you have a running Jupyter Notebooks server that you can access through a web GUI.

You can now type `exit` to detach from the `FUNKY\_NAME\_OF\_CONTAINER` container. If you're planning on using the `miniconda3` image for this series (as opposed to `anaconda3`) then you'll need to `docker exec` into the running container from time to time to install packages using the `conda` package manager. If you end up needing to restart the docker container make note that the container will get a new randomly-generated name so you will have to run `docker container ls` again to find out what it is.

# Further Work

If you've used Docker before you know that you can wrap existing Docker images with your own `Dockerfile` to enhance them. There's nothing stopping you from making your own custom Dockerfile which wraps `miniconda3` with the Jupyter installation commands above using `RUN` and provide a `CMD` which starts Jupyter automatically. This will also provide you the opportunity to give the container a consistent name so you're not running `docker container ls` all the time to figure it out.

I leave this exploration of Docker to the reader, but I guarantee you will benefit from learning about container-based development environments. Again, I highly recommend the [Dive into Docker](https://diveintodocker.com) course but feel free to find your own introduction as well.

See you next time!