

Mastering Machine Learning with Python in Six Steps

A Practical Implementation Guide to
Predictive Data Analytics Using Python

—
Second Edition

—
Manohar Swamynathan

Apress®

Mastering Machine Learning with Python in Six Steps

**A Practical Implementation Guide
to Predictive Data Analytics
Using Python**

Second Edition

Manohar Swamynathan

Apress®

Mastering Machine Learning with Python in Six Steps

Manohar Swamynathan
Bangalore, Karnataka, India

ISBN-13 (pbk): 978-1-4842-4946-8

ISBN-13 (electronic): 978-1-4842-4947-5

<https://doi.org/10.1007/978-1-4842-4947-5>

Copyright © 2019 by Manohar Swamynathan

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr

Acquisitions Editor: Celestin Suresh John

Development Editor: James Markham

Coordinating Editor: Aditee Mirashi

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-4946-8. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author	ix
About the Technical Reviewer	xi
Acknowledgments	xiii
Introduction	xv
Chapter 1: Step 1: Getting Started in Python 3.....	1
The Best Things in Life Are Free	1
The Rising Star.....	3
Choosing Python 2.x or Python 3.x.....	3
Windows.....	5
OSX.....	5
Linux.....	6
From Official Website	6
Running Python	6
Key Concepts	7
Python Identifiers	7
Keywords.....	7
My First Python Program	8
Code Blocks.....	8
Basic Object Types.....	10
When to Use List, Tuple, Set, or Dictionary	13
Comments in Python	14
Multiline Statements	14
Multiple Statements on a Single Line.....	15
Basic Operators	15
Control Structures	24

TABLE OF CONTENTS

- Lists 29
- Tuples 33
- Sets 37
- Dictionary 45
- User-Defined Functions 51
- Modules 55
- File Input/Output..... 57
- Opening a File..... 58
- Exception Handling..... 59
- Summary..... 64
- Chapter 2: Step 2: Introduction to Machine Learning..... 65**
- History and Evolution 66
- Artificial Intelligence Evolution..... 70
 - Different Forms..... 71
- Machine Learning Categories 82
 - Supervised Learning..... 83
 - Unsupervised Learning..... 84
 - Reinforcement Learning..... 84
- Frameworks for Building ML Systems 85
 - Knowledge Discovery in Databases 86
 - Cross-Industry Standard Process for Data Mining 88
- SEMMA (Sample, Explore, Modify, Model, Assess) 91
 - Sample 91
 - Explore..... 91
 - Modify..... 91
 - Model..... 91
 - Assess 92
- Machine Learning Python Packages 93
 - Data Analysis Packages..... 94
 - Machine Learning Core Libraries..... 142
- Summary..... 143

Chapter 3: Step 3: Fundamentals of Machine Learning	145
Machine Learning Perspective of Data	145
Scales of Measurement.....	146
Feature Engineering.....	149
Dealing with Missing Data.....	150
Handling Categorical Data	150
Normalizing Data	152
Feature Construction or Generation	154
Supervised Learning–Regression	163
Correlation and Causation	165
Fitting a Slope	166
How Good Is Your Model?	168
Polynomial Regression	173
Multivariate Regression.....	179
Regularization.....	194
Nonlinear Regression	198
Supervised Learning–Classification.....	199
Logistic Regression	200
Evaluating a Classification Model Performance	205
ROC Curve	207
Fitting Line.....	208
Stochastic Gradient Descent	210
Regularization.....	211
Multiclass Logistic Regression	214
Supervised Learning–Process Flow	219
Decision Trees	220
Support Vector Machine	226
k-Nearest Neighbors	230
Time-Series Forecasting	233

TABLE OF CONTENTS

- Unsupervised Learning Process Flow 244
 - Clustering 245
 - Principal Component Analysis (PCA)..... 257
- Summary..... 261
- Chapter 4: Step 4: Model Diagnosis and Tuning 263**
 - Optimal Probability Cutoff Point..... 264
 - Which Error Is Costly? 268
 - Rare Event or Imbalanced Dataset..... 268
 - Which Resampling Technique Is the Best? 272
 - Bias and Variance 274
 - Bias..... 274
 - Variance..... 274
 - K-Fold Cross Validation 276
 - Stratified K-fold Cross-Validation 277
 - Ensemble Methods 280
 - Bagging 281
 - Feature Importance 284
 - RandomForest 285
 - Extremely Randomized Trees (ExtraTree) 285
 - How Does the Decision Boundary Look?..... 286
 - Bagging—Essential Tuning Parameters 289
 - Boosting 289
 - Example Illustration for AdaBoost 290
 - Gradient Boosting 295
 - Boosting—Essential Tuning Parameters 298
 - Xgboost (eXtreme Gradient Boosting)..... 299
 - Ensemble Voting—Machine Learning’s Biggest Heroes United 304
 - Stacking..... 308

Hyperparameter Tuning.....	312
GridSearch.....	312
RandomSearch.....	314
Bayesian Optimization.....	316
Noise Reduction for Time-Series IoT Data.....	319
Summary.....	322
Chapter 5: Step 5: Text Mining and Recommender Systems.....	325
Text Mining Process Overview.....	326
Data Assemble (Text).....	327
Social Media.....	329
Data Preprocessing (Text).....	334
Convert to Lower Case and Tokenize.....	334
Removing Noise.....	336
Part of Speech (PoS) Tagging.....	338
Stemming.....	340
Lemmatization.....	342
N-grams.....	345
Bag of Words.....	347
Term Frequency-Inverse Document Frequency (TF-IDF).....	350
Data Exploration (Text).....	351
Frequency Chart.....	352
Word Cloud.....	353
Lexical Dispersion Plot.....	354
Cooccurrence Matrix.....	355
Model Building.....	356
Text Similarity.....	357
Text Clustering.....	359
Topic Modeling.....	364
Text Classification.....	367
Sentiment Analysis.....	369
Deep Natural Language Processing (DNLP).....	371

TABLE OF CONTENTS

- Word2Vec 373
- Recommender Systems 375
 - Content-Based Filtering..... 376
 - Collaborative Filtering (CF) 377
- Summary..... 381
- Chapter 6: Step 6: Deep and Reinforcement Learning 383**
- Artificial Neural Network (ANN)..... 385
 - What Goes On Behind, When Computers Look at an Image? 386
 - Perceptron—Single Artificial Neuron 387
 - Multilayer Perceptrons (Feedforward Neural Network) 390
 - Restricted Boltzman Machines (RBMs) 396
 - MLP Using Keras 402
 - Autoencoders 407
 - Convolutional Neural Network (CNN)..... 414
 - CNN on MNIST Dataset..... 423
 - Recurrent Neural Network (RNN)..... 428
 - Transfer Learning 433
- Reinforcement Learning..... 438
- Summary..... 442
- Chapter 7: Conclusion..... 443**
- Tips 445
 - Start with Questions/Hypothesis, Then Move to Data! 445
 - Don't Reinvent the Wheel from Scratch..... 446
 - Start with Simple Models 447
 - Focus on Feature Engineering..... 447
 - Beware of Common ML Imposters 448
- Happy Machine Learning 448
- Index..... 449**

About the Author



Manohar Swamynathan is a data science practitioner and an avid programmer with over 14 years of experience in various data science related areas that include: data warehousing, business intelligence (BI), analytical tool development, ad-hoc analysis, predictive modeling, data science product development, consulting, formulating strategy and executing analytics programs. He's had a career covering the life cycle of data across different domains such as US mortgage banking, retail/e-commerce, insurance, and industrial IoT. He has a bachelor's degree with a specialization in physics, mathematics, and computers; and a master's degree in project management. He's currently living in Bengaluru, the silicon valley of India.

He's also involved in technical review of books on data science using Python and R.

About the Technical Reviewer



Jojo Moolayil is an artificial intelligence professional and published author of three books on machine learning, deep learning, and IoT. He is currently working with Amazon Web Services as a Research Scientist – AI in their Vancouver, BC office.

He was born and raised in Pune, India and graduated from the University of Pune with a major in information technology engineering. His passion for problem solving and data-driven decision making led him to start a career with **Mu Sigma Inc.**, the world’s largest pure play analytics provider. Here, he was responsible for developing machine learning and decision science solutions for large complex problems for healthcare and telecom giants. He later worked with **Flutura** (an IoT Analytics startup) and **General Electric** with a focus on industrial AI in Bangalore, India.

In his current role with AWS, he works on researching and developing large-scale AI solutions for combating fraud and enriching customers’ payment experience in the cloud. He is also actively involved as a tech reviewer and AI consultant with leading publishers, and has reviewed more than a dozen books on machine learning, deep learning, and business analytics.

You can reach out to Jojo at

- www.jojomoolayil.com/
- www.linkedin.com/in/jojo62000
- <https://twitter.com/jojo62000>

Acknowledgments

I'm grateful to my mom, dad, and loving brother. I thank my wife Usha and son Jivin for providing me the space to write this book. I would like to express my gratitude to my colleagues/friends from current/previous organizations for their inputs, inspiration, and support. Thanks to Jojo for the encouragement to write this book and his technical review inputs. Big thanks to the Apress team for their constant support and help.

I would like to express my gratitude for the encouragement received from Ajit Jaokar.

Thanks for the input, feedback both positive and constructive provided by readers of the first edition of this book.

Finally, I would like to thank YOU, for showing an interest in this book and I sincerely hope to help your machine learning quest.

Introduction

This book is your practical guide to moving from novice to master in machine learning (ML) with Python 3 in six steps. The six steps path has been designed based on the “six degrees of separation” theory, which states that everyone and everything is a maximum of six steps away. Note that the theory deals with the *quality* of connections, rather than their existence. So a great effort has been taken to design an eminent yet simple six steps covering fundamentals to advanced topics gradually, to help a beginner walk his/her way from no or least knowledge of ML in Python all the way to becoming a master practitioner. This book is also helpful for current ML practitioners to learn advanced topics such as hyperparameter tuning, various ensemble techniques, natural language processing (NLP), deep learning, and the basics of reinforcement learning.

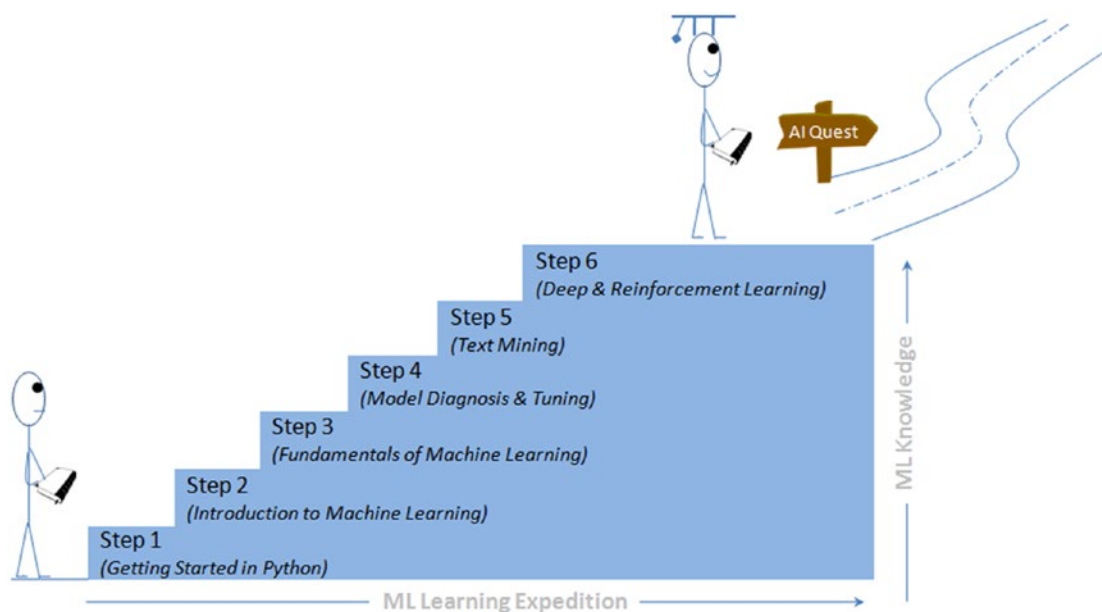


Figure 1. Mastering machine learning with Python 3 in six steps

INTRODUCTION

Each topic has two parts: the first part will cover the theoretical concepts and the second part will cover practical implementation with different Python packages. The traditional approach of math to ML (i.e., learning all the mathematics then understanding how to implement them to solve problems) needs a great deal of time/effort, which has proved to be inefficient for working professionals looking to switch careers. Hence, the focus in this book has been more on simplification, such that the theory/math behind algorithms have been covered only to the extent required to get you started.

I recommend that you work with the book instead of reading it. Real learning goes on only through active participation. Hence, all the code presented in the book is available in the form of Jupyter notebooks to enable you to try out these examples yourselves and extend them to your advantage or interest as required later.

Who This Book Is For

This book will serve as a great resource for learning ML concepts and implementation techniques for:

- Python developers or data engineers looking to expand their knowledge or career into the machine learning area
- Current non-Python (R, SAS, SPSS, Matlab or any other language) ML practitioners looking to expand their implementation skills in Python
- Novice ML practitioners looking to learn advanced topics such as hyperparameter tuning, various ensemble techniques, natural language processing (NLP), deep learning, and the basics of reinforcement learning

What You Will Learn

Chapter 1, *Step 1: Getting Started in Python 3* will help you to set up the environment, and introduce you to the key concepts of Python 3 programming language relevant to machine learning. If you are already well versed in Python 3 basics, I recommend you to glance through the chapter quickly and move on to the next chapter.

Chapter 2, *Step 2: Introduction to Machine Learning*: Here you will learn about the history, evolution and different frameworks in practice for building ML systems.

I think this understanding is very important, as it will give you a broader perspective and set the stage for your further expedition. You'll understand the different types of ML (supervised/unsupervised/reinforcement learning). You will also learn the various concepts involved in core data analysis packages (NumPy, Pandas, Matplotlib) with example codes.

Chapter 3, *Step 3: Fundamentals of Machine Learning*: This chapter will expose you to various fundamental concepts involved in feature engineering, supervised learning (linear regression, nonlinear regression, logistic regression, time series forecasting, and classification algorithms), and unsupervised learning (clustering techniques, dimension reduction technique) with the help of Scikit-learn and statsmodel packages.

Chapter 4, *Step 4: Model Diagnosis and Tuning*: in this chapter you'll learn advanced topics around different model diagnosis, which covers the common problems that arise and various tuning techniques to overcome these issues to build efficient models. The topics include choosing the correct probability cutoff, handling an imbalanced dataset, the variance, and the bias issues. You'll also learn various tuning techniques such as ensemble models, and hyperparameter tuning using grid/random search.

Chapter 5, *Step 5: Text Mining and Recommender Systems*: Statistics say 70% of the data available in the business world is in the form of text, so text mining has vast scope across various domains. You will learn the building blocks and basic concepts to advanced NLP techniques. You'll also learn the recommender systems that are most commonly used to create personalization for customers.

Chapter 6, *Step 6: Deep and Reinforcement Learning*: There has been a great advancement in the area of artificial neural networks (ANNs) through deep learning techniques, and it has been the buzzword in recent times. You'll learn various aspects of deep learning such as multilayer perceptrons, convolutional neural networks (CNNs) for image classification, RNNs (recurrent neural network) for text classification, and transfer learning. You'll also use a Q-learning example to understand the concept of reinforcement learning.

Chapter 7, *Conclusion*: This chapter summarizes your six-step learning and includes quick tips that you should remember while starting with real-world machine learning problems.

Note An appendix covering Generative Adversarial Networks (GAN) is available as part of this book's source code package, which can be accessed via the **Download Source Code** button located at www.apress.com/9781484249468.

CHAPTER 1

Step 1: Getting Started in Python 3

In this chapter you will get a high-level overview about Python language and its core philosophy, how to set up the Python 3 development environment, and the key concepts around Python programming to get you started with basics. This chapter is an additional step or the prerequisite step for nonPython users. If you are already comfortable with Python, I would recommend you to quickly run through the contents to ensure you are aware of all the key concepts.

The Best Things in Life Are Free

It's been said that “*The best things in life are free!*” Python is an open source, high-level, object-oriented, interpreted, and general purpose dynamic programming language. It has a community-based development model. Its core design theory accentuates code readability, and its coding structure enables programmers to articulate computing concepts in fewer lines of code compared with other high-level programming languages such as Java, C, or C++.

The design philosophy of Python is well summarized by the document “The Zen of Python” (Python Enhancement Proposal, information entry number 20), which includes mottos such as:

- Beautiful is better than ugly—be consistent.
- Complex is better than complicated—use existing libraries.
- Simple is better than complex—keep it simple, stupid (KISS).
- Flat is better than nested—avoid nested ifs.

- Explicit is better than implicit—be clear.
- Sparse is better than dense—separate code into modules.
- Readability counts—indent for easy readability.
- Special cases aren't special enough to break the rules—everything is an object.
- Errors should never pass silently—use good exception handling.
- Although practicality beats purity—if required, break the rules.
- Unless explicitly silenced—use error logging and traceability.
- In ambiguity, refuse the temptation to guess—Python syntax is simpler; however, many times we might take a longer time to decipher.
- Although the way may not be obvious at first—there is not only one way of achieving something.
- There should be, preferably, only one obvious way to do it—use existing libraries.
- If the implementation is hard to explain, it's a bad idea—if you can't explain in simple terms, then you don't understand it well enough.
- Now is better than never—there are quick/dirty ways to get the job done rather than trying too much to optimize.
- Although never is often better than right now—although there is a quick/dirty way, don't head on a path that will not allow a graceful way back.
- Namespaces are one honking great idea, so let's do more of those! Be specific.
- If the implementation is easy to explain, it may be a good idea—simplicity is good.

The Rising Star

Python was officially born on February 20, 1991, with version number 0.9.0. Its application cuts across various areas such as website development, mobile apps development, scientific and numeric computing, desktop GUI, and complex software development. Even though Python is a more general-purpose programming and scripting language, it has gained popularity over the past couple of years among data engineers, scientists, and Machine Learning (ML) enthusiasts.

There are well-designed development environments such as Jupyter Notebook and Spyder that allow for a quick examination of the data and enable developing of ML models interactively.

Powerful modules such as NumPy and Pandas exist for the efficient use of numeric data. Scientific computing is made easy with the SciPy package. A number of primary ML algorithms have been efficiently implemented in scikit-learn (also known as sklearn). HadoopPy and PySpark provide a seamless work experience with big data technology stacks. Cython and Numba modules allow executing Python code on par with the speed of C code. Modules such as nosetest emphasize high quality, continuous integration tests, and automatic deployment.

Combining all of these has led many ML engineers to embrace Python as the choice of language to explore data, identify patterns, and build and deploy models to the production environment. Most importantly, the business-friendly licenses for various key Python packages are encouraging the collaboration of businesses and the open source community for the benefit of both worlds. Overall, the Python programming ecosystem allows for quick results and happy programmers. We have been seeing the trend of developers being part of the open source community to contribute to the bug fixes and new algorithms for use by the global community, at the same time protecting the core IP of the respective company they work for.








Choosing Python 2.x or Python 3.x

Python version 3.0, released in December 2008, is backward incompatible. That's because as there was big stress from the development team stressed separating binary data from textual data, and making all textual data automatically support Unicode so that project teams can work with multiple languages easily. As a result, any project

migration from 2.x to 3.x required large changes. Python 2.x originally had a scheduled end-of-life (EOL) for 2015 but was extended for another 5 years to 2020.

Python 3 is a cutting edge, nicer and more consistent language. It is the future of the Python language and it fixes many of the problems that are present in Python 2. Table 1-1 shows some of the key differences.

Table 1-1. Python 2 vs. Python 3

Python 2	Python 3
 It'll retire by 2020; till then it'll receive updates for security and bug fixes.	 It has seen great adoption in the last two years; currently, 99.7% of key packages support Python 3.
 =  <p>Print is a statement. Print "Hello World!"</p>	 = $f(x)$ <p>Print is a function. Print ("Hello World!")</p>
<p>ASCII Strings are by default stored as ASCII.</p>  Rounds the integer division to the nearest whole number	<p>UNICODE Strings are by default stored as Unicode.</p>  Integer division returns the exact value without rounding to the nearest whole number.

As of now, Python 3 readiness (<http://py3readiness.org/>) shows that 360 of the 360 top packages for Python support 3.x. It is highly recommended that we use Python 3.x for development work.

I recommend Anaconda (Python distribution), BSD licensed, which gives you permission to use it commercially and for redistribution. It has around 474 packages, including the most important for most scientific applications, data analysis, and ML such as NumPy, SciPy, Pandas, Jupyter Notebook, matplotlib, and scikit-learn. It also provides a superior environment tool, conda, which allows you to easily switching between environments—even between Python 2 and 3 (if required). It is also updated very quickly as soon as a new version of a package is released; you can just do `conda update <packagename>` to update it.

You can download the latest version of Anaconda from their official website <https://www.anaconda.com/distribution/> and follow the installation instructions.

To install Python, refer to the following sections.

Windows

1. Download the installer, depending on your system configuration (32 or 64 bit).
2. Double-click the .exe file to install Anaconda and follow the installation wizard on your screen.

OSX

For Mac OS, you can install either through the graphical installer or from the command line.

Graphical Installer

1. Download the graphical installer.
2. Double-click the downloaded .pkg file and follow the installation wizard instructions on your screen.

Command Line Installer

1. Download the command-line installer
2. In your terminal window, type and follow the instructions: `bash <Anaconda3-x.x.x-MacOSX-x86_64.sh>`

Linux

1. Download the installer, depending on your system configuration.
2. In your terminal window, type and follow the instructions: `bash Anaconda3-x.x.x-Linux-x86_xx.sh`.

From Official Website

If you don't want to go with the Anaconda build pack, you can go to Python's official website www.python.org/downloads/ and browse to appropriate OS section and download the installer. Note that OSX and most of the Linux comes with preinstalled Python, so there is no need for additional configuring.

When setting up a PATH for Windows, make sure to check the "Add Python to PATH option," when you run the installer. This will allow you to invoke Python interpreter from any directory.

If you miss ticking the "Add Python to PATH option," follow these steps:

1. Right click "My computer"
2. Click "Properties"
3. Click "Advanced system settings" in the side panel
4. Click "Environment Variables"
5. Click "New" below system variables.
6. In name, enter `pythonexe` (or anything you want).
7. In value, enter the path to your Python (example: `C:\Python32\`).
8. Now edit the Path variable (in the system part) and add `%pythonexe%`; to the end of what's already there.

Running Python

From the command line, type "Python" to open the interactive interpreter. A Python script can be executed at the command line using the syntax

```
python <scriptname.py>.
```

Key Concepts

There are many fundamental concepts in Python, and understanding them is essential for you to get started. The remainder of the chapter takes a concise look at them.

Python Identifiers

As the name suggests, identifiers help us to differentiate one entity from another. Python entities such as class, functions, and variables are called identifiers.

- It can be a combination of upper or lower case letters (a to z or A to Z).
- It can be any digits (0 to 9) or an underscore (_).
- The general rules to be followed for writing identifiers in Python:
 - It cannot start with a digit. For example, 1variable is not valid, whereas variable1 is valid.
 - Python reserved keywords (refer to Table 1-2) cannot be used as identifiers.
 - Except for underscore (_), special symbols like !, @, #, \$, %, etc. cannot be part of the identifiers.

Keywords

Table 1-2 lists the set of reserved words used in Python to define the syntax and structure of the language. Keywords are case sensitive, and all the keywords are in lowercase except *True*, *False*, and *None*.

Table 1-2. Python Keywords

False	class	finally	Is	return
None	continue	for	lambda	try
True	Def	From	nonlocal	while
and	Del	Global	Not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

My First Python Program

Working with Python is comparatively a lot easier than other programming languages (Figure 1-1). Let's look at how an example of executing a simple print statement can be done in a single line of code. You can launch the Python interactive on the command prompt, type the following text, and press Enter.

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

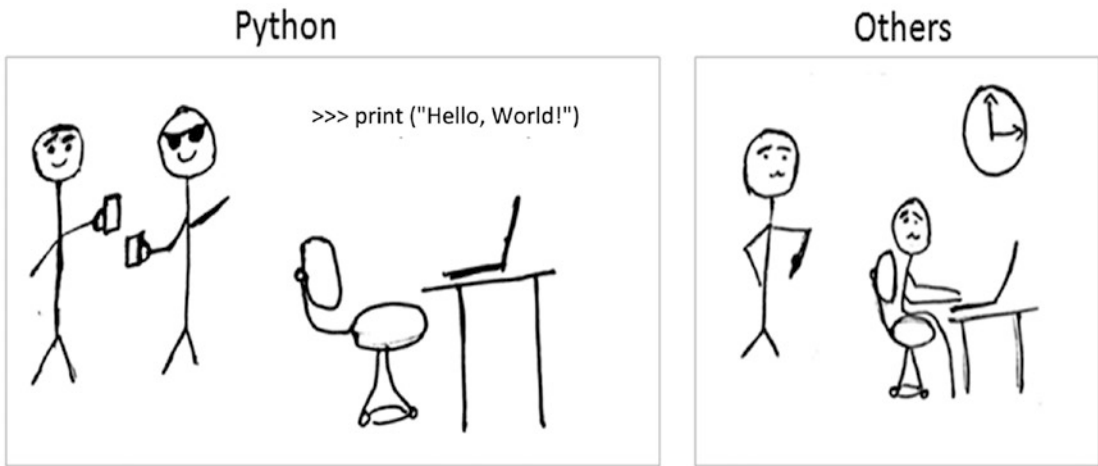


Figure 1-1. Python vs. others

Code Blocks

It is very important to understand how to write code blocks in Python. Let's look at two key concepts around code blocks: indentations and suites.

Indentations

One of the most unique features of Python is its use of indentation to mark blocks of code. Each line of code must be indented by the same amount to denote a block of code in Python. Unlike most other programming languages, indentation is not used to help make the code look pretty. Indentation is required to indicate which block of code or statement belongs to current program structure (see Listings 1-1 and 1-2 for examples).

Suites

A collection of individual statements that makes a single code block are called suites in Python. A header line followed by a suite is required for compound or complex statements such as `if`, `while`, `def`, and `class` (we will understand each of these in detail in the later sections). Header lines begin with a keyword, and terminate with a colon (`:`) and are followed by one or more lines that make up the suite.

Listing 1-1. Example of Correct Indentation

```
# Correct indentation
print ("Programming is an important skill for Data Science")
print ("Statistics is an important skill for Data Science")
print ("Business domain knowledge is an important skill for Data Science")

# Correct indentation, note that if statement here is an example of suites
x = 1
if x == 1:
    print ('x has a value of 1')
else:
    print ('x does NOT have a value of 1')
```

Listing 1-2. Example of Incorrect Indentation

```
# incorrect indentation, program will generate a syntax error
# due to the space character inserted at the beginning of the second line
print ("Programming is an important skill for Data Science")
    print ("Statistics is an important skill for Data Science")
print ("Business domain knowledge is an important skill for Data Science")
3

# incorrect indentation, program will generate a syntax error
# due to the wrong indentation in the else statement
x = 1
if x == 1:
```



```
print ('x has a value of 1')
else:
print ('x does NOT have a value of 1')
-----Output-----
print ("Statistics is an important skill for Data Science")
^
```

IndentationError: unexpected indent

Basic Object Types

Table 1-3 lists the Python object types. According to the Python data model reference, objects are Python’s notion for data. All data in a Python program is represented by objects or by relations between objects. In a sense, and in conformance to Von Neumann’s model of a “stored program computer,” code is also represented by objects.

Every object has an identity, a type, and a value. Listing 1-3 provides example code to understand object types.

Table 1-3. *Python Object Types*

Type	Examples	Comments
None	None	# singleton null object
Boolean	True, False	
Integer	-1, 0, 1, sys.maxint	
Long	1L, 9787L	
Float	3.141592654	
	inf, float('inf')	# infinity
	-inf	# neg infinity
	nan, float('nan')	# not a number
Complex	2+8j	# note use of j
String	'this is a string', "also me" r'raw string', u'unicode string'	# use single or double quote
Tuple	empty = () (1, True, 'ML')	# empty tuple # immutable list or unalterable list
List	empty = [] [1, True, 'ML']	empty list # mutable list or alterable list
Set	empty = set() set(1, True, 'ML')	# empty set # mutable or alterable
dictionary	empty = {} {'1':'A', '2':'AA', True = 1, False = 0}	# mutable object or alterable object
File	f = open('filename', 'rb')	

Listing 1-3. Code for Basic Object Types

```

none = None           #singleton null object
boolean = bool(True)
integer = 1
Long = 3.14

# float
Float = 3.14
Float_inf = float('inf')
Float_nan = float('nan')

# complex object type, note the usage of letter j
Complex = 2+8j

# string can be enclosed in single or double quote
string = 'this is a string'
me_also_string = "also me"

List = [1, True, 'ML'] # Values can be changed
Tuple = (1, True, 'ML') # Values can not be changed
Set = set([1,2,2,2,3,4,5,5]) # Duplicates will not be stored

# Use a dictionary when you have a set of unique keys that map to values
Dictionary = {'a':'A', 2:'AA', True:1, False:0}

# lets print the object type and the value
print (type(none), none)
print (type(boolean), boolean)
print (type(integer), integer)
print (type(Long), Long)
print (type(Float), Float)
print (type(Float_inf), Float_inf)
print (type(Float_nan), Float_nan)
print (type(Complex), Complex)
print (type(string), string)
print (type(me_also_string), me_also_string)
print (type(Tuple), Tuple)

```

```

print (type(List), List)
print (type(Set), Set)
print (type(Dictionary), Dictionary)

----- output -----

<type 'NoneType'> None
<type 'bool'> True
<type 'int'> 1
<type 'float'> 3.14
<type 'float'> 3.14
<type 'float'> inf
<type 'float'> nan
<type 'complex'> (2+8j)
<type 'str'> this is a string
<type 'str'> also me
<type 'tuple'> (1, True, 'ML')
<type 'list'> [1, True, 'ML']
<type 'set'> set([1, 2, 3, 4, 5])
<type 'dict'> {'a': 'A', True: 1, 2: 'AA', False: 0}

```

When to Use List, Tuple, Set, or Dictionary

Four key, commonly used Python objects are list, tuple, set, and dictionary. It's important to understand when to use these, to be able to write efficient code.

- *List*: Use when you need an ordered sequence of homogenous collections whose values can be changed later in the program.
- *Tuple*: Use when you need an ordered sequence of heterogeneous collections whose values need not be changed later in the program.
- *Set*: It is ideal for use when you don't have to store duplicates and you are not concerned about the order of the items. You just want to know whether a particular value already exists or not.
- *Dictionary*: It is ideal for use when you need to relate values with keys, in order to look them up efficiently using a key.

Comments in Python

Single line comment: Any characters followed by the # (hash) and up to the end of the line are considered as part of the comment and the Python interpreter ignores them.

Multiline comments: Any characters between the strings `"""` (referred to as multiline string), that is, one at the beginning and end of your comments, will be ignored by the Python interpreter. Please refer to Listing 1-4 for a comments code example.

Listing 1-4. Example Code for Comments

```
# This is a single line comment in Python
print("Hello Python World") # This is also a single line comment in Python

""" This is an example of a multi-line
the comment that runs into multiple lines.
Everything that is in between is considered as comments
"""
```

Multiline Statements

Python's oblique line continuation inside parentheses, brackets, and braces is the favorite way of casing longer lines. Using a backslash to indicate line continuation makes readability better; however, if needed you can add an extra pair of parentheses around the expression. It is important to indent the continued line of your code suitably. Note that the preferred place to break around the binary operator is after the operator, and not before it. Please refer to Listing 1-5 for Python code examples.

Listing 1-5. Example Code for Multiline Statements

```
# Example of implicit line continuation
x = ('1' + '2' +
     '3' + '4')

# Example of explicit line continuation
y = '1' + '2' + \
    '11' + '12'

weekdays = ['Monday', 'Tuesday', 'Wednesday',
             'Thursday', 'Friday']
```

```

weekend = {'Saturday',
           'Sunday'}

print ('x has a value of', x)
print ('y has a value of', y)
print (weekdays)
print (weekend)

----- output -----
('x has a value of', '1234')
('y has a value of', '1234')
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
set(['Sunday', 'Saturday'])

```

Multiple Statements on a Single Line

Python also allows multiple statements on a single line through the usage of the semicolon (;), given that the statement does not start a new code block. Listing 1-6 provides a code example.

Listing 1-6. Code Example for Multiple Statements on a Single Line

```
import os; x = 'Hello'; print (x)
```

Basic Operators

In Python, operators are the special symbols that can manipulate the value of operands. For example, let's consider the expression $1 + 2 = 3$. Here, 1 and 2 are called operands, which are the value on which operators operate, and the symbol + is called operator.

Python language supports the following types of operators:

- Arithmetic operators
- Comparison or Relational operators
- Assignment operators
- Bitwise operators
- Logical operators