Contents

[1 BASIC PYTHON FOUNDATION COURSE 2](#_Toc12276621)

[2 INTERMEDIATE PYTHON TOPICS 22](#_Toc12276622)

[3 ADVANCED PYTHON TOPICS 24](#_Toc12276623)

# INTRODUCTION

# IP

As an author, I will always strive to cite sources.

# BASIC PYTHON FOUNDATION COURSE

| **MAIN TOPIC** | **SUB TOPIC** | **DETAILS**  **/**  **(LINKS FOR FURTHER STUDY)**  **/**  **(FEEDBACK)**  **/**  **(SAMPLE PROGRAMS)** | **(CLASSROOM EXERCISES)**  **/**  **(ASSIGNMENTS)** | **TRACKING INFORMATION** |
| --- | --- | --- | --- | --- |
| **OVERALL CONTEXT** | WHAT ARE YOU EXPECTING ? | <Update after feedback from the students> | **N/A** | DAY 1  (<=15 mins) |
|  | MY EXPECTATIONS FROM THE STUDENTS/YOU | * Be aware of the course content (*Have all of you gone through the course details [separate doc] ?)* * Do the class room exercises * Complete your assignments * Make notes *(I do it and it helps me)* * Don’t just nod your head to what I say. Digest it slowly. Stop me if I am going too fast | **N/A** | DAY 1  (<= 15 mins) |
|  | GOALS AS A TRAINER | * Teach the basics of Python in an informal and interactive way * Ensure that the course content is covered within the given time frame * Get you motivated to start programming * Get you curious about the language. Python is real fun to program with !. * Ensure that you are generally attentive during the course * Get positive feedback at the end of the course | **N/A** | DAY 1  (<= 15 mins) |
|  | PROPOSED TEACHING STYLE | * Interactive / personalized style with lots of visuals   i.e. why I use a WORD document instead of a PPT   * Avoid the use of lengthy paragraphs * Avoid spending too much time talking on a specific topic * Switching contexts too often – Strict NO NO * Get all of you to write lots of programs * Will try to use images, because our brain understands visuals more easily than text | **N/A** | DAY 1  (<= 15 mins) |
|  | ABOUT THE TRAINER I.E MYSELF |  | **N/A** | DAY 1  (<= 15 mins) |
|  | NEXT STEPS | We will revisit this topic at the end of the course | **N/A** | DAY 1  (<= 15 mins) |
|  | ANY OTHER POINTS ? | * <Update after feedback from the students> | **N/A** | DAY 1  (<= 15 mins) |
| **PYTHON ECOSYSTEM** | PYTHON INSTALLATION related information | * For the course, we will be using Python 3.7.x * Download link <https://www.python.org/downloads/> * Find out if there are any * Mac users * Linux users |  | DAY 1  (<= 30 mins) |
|  | Environment variables that need to be set | * Windows   + PYTHON\_HOME   This needs to point to the directory where Python 3.7.1 has been installed  *(> echo %PYTHON\_HOME%)*   * + Path      * + PYTHON\_HOME   + PYTHON\_HOME\Scripts |  | DAY 1  (<= 10 mins) |
|  | Installing/Using third party editors | * IDLE (Python’s default editor)   Comes pre-installed with Python. We will use this along with Jupyter Notebooks.  *%PYTHON\_HOME%\ \Lib\idlelib\idle.bat*   * Jupyter Notebooks (<http://jupyter.org/> )  This is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. python -m pip install --upgrade pip  pip install jupyter   * Notepad++ link – [optional]   <https://notepad-plus-plus.org/download/v7.5.9.html>   * WingWare Link [optional]   <https://wingware.com/downloads/wingide-101>   * PyCharm   <https://www.jetbrains.com/pycharm/download/#section=windows> [optional] |  | DAY 1  (<= 30 mins) |
|  | Brief history of Python | * Sources of information *(there are many)* * <https://www.python-course.eu/python3_history_and_philosophy.php> * <https://medium.com/@johnwolfe820/a-brief-history-of-python-ca2fa1f2e99e> |  | DAY 1  (<= 15 mins) |
|  | Python’s rich standard library | * The standard library is the jewel in Python’s crown, supplying reusable modules that help you with everything from, for example, working with data, through manipulating ZIP archives, to sending emails, to working with HTML. * The standard library even includes a web server, as well as the popular SQLite database technology * During the course, we will be using the standard library extensively |  | DAY 1  (<= 30 mins) |
|  | Third party modules | Python has a rich collection of third party modules that developers can use in their applications.  A few well known modules :-   * Web Development * Django * Flask * REST API development * Falcon * Flask-Restful * Data Analysis * Numpy * Pandas * Big Data (with Apache Spark) * PySpark * Machine Learning * PyTorch plus rich ecosystem * HTTP Library * Requests * Automated Unit testing * PyTest * Browser Automation * Selenium * Interactive IDEs * Jupyter * Database Toolkits * SQLAlchemy * Web scraping * Scrapy |  | DAY 1  (<= 30 mins) |
|  | The Python Interpreter | * When Python is installed, the interpreter gets installed * There is no “EXE” in Python. * When you run a program, the interpreter compiles the code and executes the compiled code * Python *<<program/module name>>* |  | DAY 1  (<= 10 mins) |
|  | Python’s interactive shell | * Executing just python opens up the interactive shell * Run “python” from the command line * >>> appears |  | DAY 1  (<= 10 mins) |
|  | What are people using Py for ? | Earlier we discussed the standard library and third party modules . This gives a good indication about what Python is being used for   * Web development * Web services * Web scraping * Batch processing * Data sciences * Data processing * Machine Learning * Big data   What’s missing from the list ?   * Mobile application development ! |  | **DAY 1**  **(<= 10 mins)** |
|  | ANY OTHER POINTS ? | * Debuggers - pdb *- (Will not be covered in this course)*   + Basic info in *<< pfc\_supporting\_info\_and\_other\_stuff.docx>>* |  | **DAY 1**  **(<= 10 mins)** |
|  | Learning to use interactive environments [ Python’s default interactive shell, Jupyter] | * Used to run snippets of Python code * Generally used for executing a single statement at a time * Helpful when starting to learn Python * More info in <<py\_interactive\_shell\_the\_basics.py>> * Jupyter is major step forward and is being used extensively now. During the training course, Jupyter will be used extensively. |  | **DAY 2**  **(<=1 hr)** |
|  | Learning to use IDLE – Python’s default editor | * Launching IDLE * <Python\_Home>\Lib\idlelib\idle.bat   *(Python\_home is the environment variable)*  This will be short and sweet as the focus is on Jupyter Notebooks |  | **DAY 2**  **(<=30 mins)** |
|  | WingWare IDE | * Time permitting, we might look at this IDE later   *(There is no one size fits “ALL”. Developers use a wide variety of IDEs for developing Python apps.)*  Always choose one that you are comfortable with. |  | **DAY 2** |
| **PYTHON BASIC PROGRAMMING** | What do you think a Python program contains ? |  | Get the audience to provide inputs | **DAY 2**  **(<= 30 mins)** |
|  | In general, what do you think programs should do ? | * Handle data * Process input to generate output * Over the years, languages have evolved to manage complex data * A programming language must have data structures to be able to efficiently process information |  | **DAY 2**  **(<= 30 mins)** |
|  |  |  |  |  |
|  | Some basic code samples | *Python <program name>*   * Really\_simple \_example.py * Another\_simple\_example.py * <Create a few more at run time> | * Any ideas from the group ? | **DAY 2**  **(<= 30 mins)** |
|  | Declaring variables | * declaring\_variables\_example\_001.py * declaring\_variables\_example\_002.py * declaring\_variables\_example\_003.py * declaring\_variables\_example\_004.py * declaring\_variables\_example\_005.py * declaring\_variables\_example\_006.py * declaring\_variables\_example\_007.py   (floats)  A few points to be noted   * + The data type is not explicitly stated   + A variable can be assigned different types of values   Note – *Python programs are given a .py extension. It is not mandatory, but is an industry standard* | **PY-CL-DECLARING-VARIABLES**  Q? Why do we declare variables ?. Do we need them ?.  Python is said to be dynamically typed. Can someone explain this statement ?. | **DAY 2**  **(<= 1 hr)** |
|  | More on declaring String variables | * In most programs, strings are very widely used * Python provides extensive features for dealing with Strings * Link for further reading - <https://docs.python.org/3/library/stdtypes.html#string-methods>   Sample programs   * decl\_string\_var\_example\_001.py * String\_var\_example\_002.py   Did any of you visit the above URL ?.  Q – Can we find the frequency of a word in text ? yes  Q – Number of vowels in text ? yes | **PY-CL-DECLARING-STRING-VARIABLES**  Q?. Get the audience to give examples of Strings  Q? There are different ways to define a string value. How many?  Q? What is casting?. | **DAY 2**  **(<= 30 mins)** |
|  | Can you think of any other variables that you would like to declare in your programs ? | * Large numbers with decimals * Arrays * Multi dimensional arrays * Data structures to handle non structured information that today’s applications generate * Images |  | **DAY 2** |
|  | CLASS ROOM EXERCISES |  |  | **DAY 2** |
|  | ANY OTHER POINTS ? | <Update after feedback from students> |  | **DAY 2** |
|  | **EVERYTHING IS AN OBJECT** | * This is something that you will read about all the time * We will, over the next few weeks, whenever required, re-visit this topic * Each language has its own way of interpreting what an object actually is * More details in *<< pfc\_supporting\_info\_and\_other\_stuff.docx >>* * Sample code * py\_everything\_is\_an\_obj\_001.py * py\_everything\_is\_an\_obj\_002.py * py\_everything\_is\_an\_obj\_003.py * py\_everything\_is\_an\_obj\_004.py   This might not be fully understood at this stage.   * More sample programs for mutable types * everything\_is\_an\_obj\_example\_001.py * everything\_is\_an\_obj\_example\_002.py   [x = x + x + x + x + x + x + x and other strange stuff  1 + “x” will not work as well ]  DON”T WORRY, we will continue to discuss this important topic in later sessions. | * PY-CL-VARIABLES-OBJECTS   + Declare 5 integer variables, set the value to 1   Use id and type functions for all 5, what do you observe ?   * + Use the above program   Add the following code -  Set each of the 5 variables to “pibm”  Use id and type functions for all 5, Now, what do you observe ? | **DAY 2** |
| **THE NEXT LEVEL OF PYTHON PROGRAMS** | In your opinion what can we cover next ? | <update after feedback from students> |  | **DAY 2** |
|  | INDENTATION | * Every language has its own syntax * There is a specific way to identify statements, a group of statements and blocks of code * Examples are “;”, “{“, “}”, etc.   Python’s approach is very different and is referred to as “INDENTATION”. Please refer to *<< pfc\_supporting\_info\_and\_other\_stuff.docx>>*   * Indentation is something that is both unique and peculiar to Python * How to statements end ? * What is a block of code in Python ? * “Suite” terminology   Some examples of bad indentation   * indentation\_bad\_example\_001.py * indentation\_bad\_example\_002.py * indentation\_bad\_example\_003.py | * Get the audience to play around with bad indentation | **DAY 2** |
|  | If | * We will learn to create decisions in a Python program using the various forms of “if” * Learn to conditionally execute code in a program * Sample code - <<If\_example\_001.py>> | * Get the audience to write pseudo code with IF | **DAY2** |
|  | If else | * Sample code - <<if\_else\_example\_001.py>> * if\_else\_ladder\_example\_001.py   (Avoid writing such code !) | Get the audience to write pseudo code with IF-ELSE | **DAY 2** |
|  | Switch statement ? | Languages like Java and PHP have in-built switch statements to avoid complicated if else ladders.  The Pythonic way to implement switch statement is to use dictionary mappings (covered later in the course), that provide simple one-to-one key-value mappings.  Will be discussed when we get to Dictionaries |  |  |
|  | If elif | * Sample code - << if\_elif\_example\_001.py >> * Sample code - << if\_elif\_nesting\_example\_001.py >> | Get the audience to write pseudo code with IF | **DAY 2** |
|  | WHILE loops | * Sample code - << while\_loop\_example\_001.py >> * Sample code –   << while\_loop\_with\_else\_example\_001.py >> | Get the audience to write pseudo code with IF | **DAY 2** |
|  | FOR Loops | * This is very widely used in Python * The for loop in Python is used to iterate over a sequence (Examples - [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [string](https://www.programiz.com/python-programming/string)) or other **iterable** objects. * Iterating over a sequence is called traversal.   Let’s take a step back and discuss the concept of iteration - *<< pfc\_supporting\_info\_and\_other\_stuff.docx>>*   * Sample code - << for\_loop\_example\_001.py >> * Sample code - << for\_loop\_example\_002.py >> * Sample code - << for\_loop\_example\_003.py >> * <<for\_loop\_example\_004.py>> * for\_loop\_with\_else\_clause\_001.py | Get the audience to write pseudo code with IF | **DAY 2** |
|  |  | The RANGE function  One can generate a sequence of numbers using  the range() function.  Example - range(10) will generate numbers from 0 to 9 (10 numbers).  Start, stop and step size as range(start,stop,step size) can also be specified.  step size defaults to 1 if not provided. | *Note – Students will be given assignments to explore this function* |  |
|  | CLASSROOM EXERCISES |  |  |  |
|  | ANY OTHER POINTS TO DISCUSS |  |  |  |
|  |  |  |  |  |
| **DATA STRUCTURES** | Scratching the surface |  |  |  |
|  | Arrays | In Python, arrays are supported by the array module (part of the standard library) and needs to be imported before you start using them  Only numeric values are supported   * arrays\_in\_py\_example\_001.py | * Ask the audience to define some one dimensional arrays, two dimensional arrays |  |
|  | Tuples | **Tuple: an ordered immutable collection of objects**  A tuple is an immutable list. This means that once you assign objects to a tuple, the tuple cannot be changed under any circumstance.  It is often useful to think of a tuple as a constant list.   * tuple\_example\_001.py * tuple\_example\_002.py * tuple\_example\_003.py * tuple\_example\_004.py * tuple\_example\_005.py * tuple\_negative\_indexing.py * tuple\_nesting\_example\_005.py * tuple\_single\_element.py * tuple\_slicing.py   Q – When to use tuples ? | * After going through the examples, ask people to create their own tuples * Q? difference, if any, between a tuple and an array * Assess the understanding of mutability and immutability from the audience |  |
|  | Lists | * lists\_example\_001.py * lists\_example\_002.py * lists\_example\_003\_app\_ext\_diff.py * lists\_example\_004\_slices.py * list\_example\_005\_comprehensions.py * import\_csv\_file\_example\_001.py   A quick review of the list methods  list.append(elem) , list.insert(index, elem) , list.extend(list2), list.index(elem), list.remove(elem), list.reverse(),list.pop(index)  Q – Is the insertion order preserved ? |  |  |
|  | Dictionaries | Python's efficient key/value hash table structure is called a "dict".  Dictionaries differ from lists in that you can access items in a dictionary by a key rather than a position  Dicts store an arbitrary number of objects, each identified by a unique dictionary key. Dictionaries are often also called maps, hashmaps, lookup tables, or associative arrays. They allow the efficient lookup, insertion, and deletion of any object associated with a given key.   * dict\_example\_001\_decl.py * dict\_example\_002\_methods.py * dict\_example\_003\_comprehension.py * dict\_example\_004\_membership\_test.py * dict\_example\_005\_for\_loop.py * dict\_example\_006\_merging.py * dict\_example\_007\_more\_stuff.py * import\_csv\_file\_example\_002.py   Q – Is the insertion order preserved ? |  |  |
|  | Sets | Because sets cannot have multiple occurrences of the same element, it makes sets highly useful to efficiently remove duplicate values from a list or tuple and to perform common math operations like unions and intersections   * sets\_example\_001\_decl.py * sets\_example\_002\_comprehension.py * sets\_example\_003\_rd.py * sets\_example\_004\_operations.py |  |  |
|  | Collections in 3.7 | The collections module provides additional data structures for handling specific situations :  namedtuple(), deque, ChainMap, Counter, OrderedDict, defaultdict, UserDict, UserList, UserString  We will look at the following only -   * Counter   coll\_counters\_example\_001/2/3.py   * namedtuple()   coll\_namedtuples\_example\_001.py   * deque,   coll\_dequeue\_example\_001.py |  |  |
|  | CSV files into data structures | * import\_csv\_file\_example\_001.py * import\_csv\_file\_example\_002.py * import\_csv\_file\_into\_dict\_example\_003.py * import\_country\_states\_data\_example.py |  |  |
|  | CLASSROOM EXERCISES |  |  |  |
|  | Any other points to discuss ? |  |  |  |
| * **D.R.Y.** * **MODULARITY** * **WRITING BETTER CODE** | Using DocString for documentation |  |  |  |
|  | Functions in Python | What is DRY ?  Let’s look at some really bad programs  Benefits of using functions in Python or for that matter any other language   * Reducing duplication of code * Decomposing complex problems into simpler pieces * Improving clarity of the code * Reuse of code * Information hiding * func\_example\_001\_without\_its\_use.py * func\_example\_001\_a\_with\_its\_use.py * func\_example\_002\_without\_its\_use.py * func\_example\_002\_a\_with\_its\_use.py * func\_example\_002\_b\_with\_its\_use.py * func\_example\_003\_main\_module.py * func\_example\_003\_uses\_a\_module.py * func\_example\_003\_b\_uses\_a\_module.py * func\_example\_003\_c\_uses\_a\_module.py | Q?. Any other benefits of good programming ?. |  |
|  | Documenting your functions | >> doc\_string\_example.py |  |  |
|  | Lambda Functions | The lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name.  Lambda functions is mainly used with built-in functions like filter(), map() and reduce()   * LAMBDAS\_example\_001.py * LAMBDAS\_example\_002.py * LAMBDAS\_example\_003.py * LAMBDAS\_example\_004.py * LAMBDAS\_example\_005.py |  |  |
|  | Exception handling | Things go wrong, all the time—no matter how good your code is.   * Your run time environment may be very different from you dev/test environments. * Production data   This is not meant for handling syntax errors. Syntax errors should be fixed by developers during unit testing of programs.  These type of errors typically occur whenever syntactically correct Python code results in an error.  Raising an Exception  One can use **raise** to throw an exception if a condition occurs.   * try\_except\_example\_001.py * try\_except\_example\_001\_a.py * try\_except\_example\_002.py * try\_except\_example\_003.py   User Defined Exceptions – Requires knowledge of OOPS | Q?. Can the audience come up with some real examples ?. |  |
|  | Logging information during execution of a program | * logging\_example\_001.py * logging\_example\_002.py | Q?. Why is it useful to log information to the console or a file at run time ?  Q? Does it slowdown execution of the program |  |
|  | Coding standards | * <https://www.python.org/dev/peps/pep-0008/> * <https://www.datacamp.com/community/tutorials/pep8-tutorial-python-code> | Q?. Why do we need to write good code ?  *[Relate this to a bad movie, a horrible news article]* |  |
|  | Static code Analysis | * In addition to knowing Python, we have been emphasizing the need to write good code * There are many tools that can analyze code and assess the code quality * PyLint is one such example and is widely used   Pylint  <https://www.pylint.org/>   * SonarQube | Q? Would you declare a variable and not use it  Q? Would you declare a variable like vname and store a date value in it  Q? Inspecting your own code  Q? Inspecting the code of other programmers |  |
|  | Modules and Packages | * Modules and Packages are terms that you will frequently encounter in your “Pythonic” journey * A module is a file consisting of Python code.   + All the above examples are modules * A **package** is a just collection of **Python** modules   i.e., a **package** is a directory of **Python** modules containing an additional \_\_init\_\_.py file. | Q? What on earth is \_\_init\_\_.py ?  Q? Why do you think we would need packages |  |
|  | A quick visit back to the house of the “Standard Library” | * Earlier on, there was a grand statement that Python’s crown jewel * <https://docs.python.org/3/library/> - Check this out * Provides standardized solutions for many problems that occur in everyday programming | Q? As a programmer, would you like to write everything from scratch or whenever required use “off the shelf ready made components”  Q? Go through the link and tell us some of the functionality provided out of the box |  |
|  | Some popular modules from the standard library | * import math * import random * import sys * import datetime | Write programs to use at least 2 functions from each of the 4 modules |  |
| **OOPS IN PYTHON** |  | OOPS is a general concept that has been implemented in many languages.  In programming two things are very important – data and behavior.  Using OOPS, one can neatly and efficiently manage both.  Python’s implementation is easy to use, powerful and fairly light weight.  oops\_in\_py\_example\_001.py  oops\_in\_py\_example\_002.py  oops\_in\_py\_example\_003.py  oops\_in\_py\_example\_004.py  oops\_in\_py\_example\_005.py  oops\_in\_py\_example\_006.py   * NOTE - A detailed discussion on OOPS is not being included as part of this course. |  |  |
| **DATA STRUCTURES** | Diving below the surface | * We have covered Data Structures earlier * Python is famous for its data structures   Lets dive into its finer details   * Comprehensions in creating data structure * Mapping data (CSV files, etc, to data structures) * Nested data structures * Populating nested data structures |  |  |
|  |  |  |  |  |
| **DATA SCIENCES IN PYTHON** |  | * This is just to get you excited about Python. Will not be covered in this course. This is an advanced topic.   Brief introduction to Pandas |  |  |
|  |  | pandas\_basics\_example\_001/2/3/4/5/6.py  pandas\_series\_example\_001.py  pandas\_level2\_example\_001.py |  |  |
| **CONCLUDING THE COURSE** | What went well ? |  |  |  |
|  | So – so ..ummm |  |  |  |
|  | Really bad |  |  |  |
|  | Next steps |  |  |  |

# INTERMEDIATE PYTHON TOPICS

| **MAIN TOPIC** | **SUB TOPIC** | **DETAILS**  **/**  **(LINKS FOR FURTHER STUDY)**  **/**  **(FEEDBACK)**  **/**  **(SAMPLE PROGRAMS)** | **(CLASSROOM EXERCISES)**  **/**  **(ASSIGNMENTS)** | **TRACKING INFORMATION** |
| --- | --- | --- | --- | --- |
| OOPS | The sequel (better or worse, you decide !) |  |  |  |
|  | Object serialization and deserialization (OSAD) | * Using Marshmallow * Why is OSAD required ?. |  |  |
|  |  |  |  |  |
| Function decorators | Introduction | * This is a deep topic * Functions are first class objects in Python. * This means that they can be passed around and used like any other python object. * Functions can be passed as arguments. Returned as arguments from functions. |  |  |
|  | Passing functions as arguments |  |  |  |
|  | Nested Functions |  |  |  |
|  | Closures |  |  |  |
|  | Doing some really weird stuff |  |  |  |
|  | Functions as decorators | Playing around with functions. Creating really weird code.   * func\_with\_decorators\_001.py * func\_with\_decorators\_002.py * func\_with\_decorators\_003.py * func\_with\_decorators\_004.py * func\_with\_decorators\_005.py |  |  |
|  | Class Decorators |  |  |  |
| **Debugging Python programs** | PDB module |  |  |  |
|  |  |  |  |  |
| **Iterators and Generators** |  | * functools * itertools * operator   Note - There is a style of programming known as functional programming. Python as a language implements some aspects/features of functional programming.  The student is advised to refer to external sources to understand more about functional programming, a topic not covered in this course. |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **VIRTUAL ENVIRONMENTS** | Why are virtual environments needed ?  Creating virtual environments. | Python applications will often use packages and modules that don’t come as part of the standard library. Applications will sometimes need a specific version of a library, because the application may require that a particular bug has been fixed or the application may be written using an obsolete version of the library’s interface.  This means it may not be possible for one Python installation to meet the requirements of every application. If application A needs version 1.0 of a particular module but application B needs version 2.0, then the requirements are in conflict and installing either version 1.0 or 2.0 will leave one application unable to run.  The solution for this problem is to create a [virtual environment](mk:@MSITStore:D:\software-home\python\Doc\python373.chm::/glossary.html#term-virtual-environment), a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages. | >> Packages, Modules  [Covered in the Foundation course]  >> Distribution – Describe each of them ?  >> Jointly go through - <https://docs.python-guide.org/dev/virtualenvs/> |  |
|  |  |  |  |  |
|  | Using standard virtual environments | > python -m venv D:\workspaces\my-python-sandpit\django | * Use python -m venv to create virtual environments on your machine * Learn about activate and deactivate * Install different packages/dependencies in different environments |  |
|  | Using pipenv | * <https://realpython.com/pipenv-guide/>   This is a tool developed by Kenneth Reitz | * Go through this with the audience * Compare pipenv vs virtual env |  |
|  |  |  |  |  |
|  | Setup.py | Using setup.py instead of Pip to install a dependency |  |  |
| CPython | CPython  (This is just theory) | The default implementation of the Python programming language is Cpython. As the name suggests Cpython is written in [C language](https://www.geeksforgeeks.org/c/). Cpython compiles the python source code into intermediate bytecode, which is executed by the Cpython virtual machine. CPython is distributed with a large standard library written in a mixture of C and Python. CPython provides the highest level of compatibility with Python packages and C extension modules.  All versions of the Python language are implemented in C because CPython is the reference implementation. | As a developer, you simply need to keep in mind that, in almost all cases, your code will be compiled into intermediate bytecode using CPython.  This is more or less the default.  {{This isn’t really a topic, but information that every developer must be aware of }} |  |
| RPython | RPython  {This is just theory} | RPython is a framework for implementing interpreters and virtual machines for programming languages, especially dynamic languages.  RPython is a restricted subset of Python that is amenable to static analysis. Although there are additions to the language and some things might surprisingly work, this is a rough list of restrictions that should be considered. Note that there are tons of special cased restrictions that you’ll encounter as you go. The exact definition is “RPython is everything that our translation toolchain can accept” :) | This is just a technical topic that you need to be aware of at a very high level.  {{This isn’t really a topic, but information that every developer must be aware of }} |  |
| PyPy | PyPy  {This is just theory} | <http://doc.pypy.org/en/latest/architecture.html> | This is just a technical topic that you need to be aware of at a very high level.  {{This isn’t really a topic, but information that every developer must be aware of }} |  |
| Jython | Jython  {This is just theory} |  | This is just a technical topic that you need to be aware of at a high level.  {{This isn’t really a topic, but information that every developer must be aware of }} |  |
|  |  |  |  |  |
| **SOME REFERENCE PROJECTS** | Sample Project to process files | There are lots of sample Python projects on the Net especially GITHUB.  Developers are encouraged to clone these projects using GIT and explore the code. | Create your very own Python project to process files  Remember the three terms – Packages, Modules and Distribution |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# ADVANCED PYTHON TOPICS

| **MAIN TOPIC** | **SUB TOPIC** | **DETAILS**  **/**  **(LINKS FOR FURTHER STUDY)**  **/**  **(FEEDBACK)**  **/**  **(SAMPLE PROGRAMS)** | **(CLASSROOM EXERCISES)**  **/**  **(ASSIGNMENTS)** | **TRACKING INFORMATION** |
| --- | --- | --- | --- | --- |
| **MULTI-THREADING IN PYTHON** |  |  |  |  |
| **CONCURRENCY** |  |  |  |  |
| **Builds and Deployments** |  |  |  |  |
| **Numpy and Pandas** | Diving into the deeper details of NumPy and Pandas | This course references material from   * <https://numpy.org/doc/1.17/user/quickstart.html> * <https://pandas.pydata.org/pandas-docs/stable/> |  |  |
|  |  |  |  |  |
|  | NUMPY | Numpy primarily provides developers with an n-dimensional array object.  AND  There are numerous functions for manipulating Numpy arrays.  Numpy is a generic framework for scientific computing; it does not know anything about AI, M, Deep Learning, Data Sciences or Neural Networks.  It is your skills as a programmer to use Numpy as a generic framework for scientific computing. All the best !.  Important Concepts   * Homogenous multi-dimensional arrays * Axes/dimensions * Indexes * Shape of an array |  |  |
|  |  | Basic numpy array operations |  |  |
|  |  | Universal Functions |  |  |
|  |  | Indexing, Slicing and Iterating |  |  |
|  |  | Shape Manipulation |  |  |
|  |  | Copies and Views of numpy arrays |  |  |
|  |  | Advanced Indexing |  |  |
|  |  | Linear Algebra |  |  |
|  |  | Visualization of numpy arrays |  |  |
|  |  |  |  |  |
|  | PANDAS | Important concepts   * Data Structures * Sparse Data Structures * Series * Time Series * Time deltas * DataFrames * Head/Tail * Indexing * Containers of Arrays * IO Tools [Importing and Exporting of data] * Slicing * Missing data * Pivot Tables * Views and Copies * Attribute Access * Reshaping * Merge, Join, Concatenate * Splitting * Categorical Data * Computational Tools * Visualization * Linear Algebra (?) |  |  |
|  |  |  |  |  |
|  | Numpy Exercises | Nearest Neighbour Search |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |