| **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   + Path   + PYTHON\_HOME   *(> echo %PYTHON\_HOME%)* |  | DAY 1  (<= 10 mins) |
|  | Installing third party editors | * 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 * Data Analysis * Numpy * Pandas * Machine Learning * PyTorch * HTTP Library * Requests * Automated Unit testing * PyTest * 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) |
|  | Different editors/ IDEs | * IDLE – This is Python’s default editor * WingWare (Will not be covered in the course) * Notepad/Notepad++ * PyCharm (Will not be covered in the course) * Jupyter Notebooks (will not be covered in the course) |  | **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 *<< pibm-ips-images-other-stuff.docx>>* |  | **DAY 1**  **(<= 10 mins)** |
|  | Learning to use the interactive shell | * 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>> |  | **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)* |  | **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 ? |  |  | **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)** |
|  |  | *06/11/2018* |  |  |
|  | Some basic code samples | *Python <program name>*   * Really\_simple \_example.py * Another\_simple\_example.py * <Create a few more at run time> |  | **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   (This feature will be explained in detail when we talk about everything is an object)  Note – *Python programs are given a .py extension. It is not mandatory, but is an industry standard* | **PY-CL-DECLARING-VARIABLES** | **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** | **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 |  | **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 *<< pibm-ips-images-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 *<< pibm-ips-images-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 ?   Some bad indentation examples   * indentation\_bad\_example\_001.py * indentation\_bad\_example\_002.py * indentation\_bad\_example\_003.py |  | **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>> |  | **DAY2** |
|  | If else | * Sample code - <<if\_else\_example\_001.py>> * if\_else\_ladder\_example\_001.py   (Avoid writing such code !) |  | **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 >> |  | **DAY 2** |
|  | WHILE loops | * Sample code - << while\_loop\_example\_001.py >> * Sample code –   << while\_loop\_with\_else\_example\_001.py >> |  | **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 - *<< pibm-ips-images-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 |  | **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 |  |  |  |
|  |  | SATURDAY = 10TH NOVEMBER |  |  |
| **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 |  |  |
|  | 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 ? |  |  |
|  | 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   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   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 |  |  |
|  | 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 |  |  |
|  | 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 |  |  |
|  | Coding standards | * <https://www.python.org/dev/peps/pep-0008/> * <https://www.datacamp.com/community/tutorials/pep8-tutorial-python-code> |  |  |
|  | Static code Analysis | Pylint  <https://www.pylint.org/> |  |  |
|  | Modules and Packages |  |  |  |
|  | A quick visit back to the house of the “Standard Library” |  |  |  |
| **OOPS IN PYTHON** |  |  |  |  |
| **DATA STRUCTURES** | Diving below the surface |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **CONCLUDING THE COURSE** | What went well ? |  |  |  |
|  | So – so ..ummm |  |  |  |
|  | Really bad |  |  |  |
|  | Next steps |  |  |  |