INDUSTRIAL TRAINING REPORT
The industrial report should be written in the following format:

Project Report Format
1. Title Page
2. Certificate Copy
3. Acknowledgement
4. Abstract
5. Contents
6. Results
7. Summary and conclusions
8. References
9. Appendices

Abstract

An abstract is a brief condensed statement by the developer. The abstract must not exceed 600 words in length and should a statement of the problem, an explanation of the methods and procedures used in gathering data, and a summary of the findings. **It should not be just a summary statement of each chapter.**

Acknowledgement

In the "Acknowledgements" page, the writer recognizes his indebtedness for guidance and assistance of the thesis adviser and other members of the faculty. Courtesy demands that he also recognize specific contributions by other persons or institutions such as libraries and research foundations. Acknowledgements should be expressed simply, tastefully, and tactfully **duly singed above the name. E-mail should also be given at the end.**

Guidelines for project report/ dissertation/thesis writing

Good quality white A4 size good quality paper should be used for typing

Page Specification

Left Margin : 3.5 cms

Right Margin : 3.0 cms

Top Margin : 2.54 cms/ 1 inch

Bottom Margin : 2.54 cms/ 1 inch

Page numbers –1. Title page should not be numbered.

2. Numbering will start from declaration till contents starting from (ii)

3. All text pages as well as program source code listings should be numbered at the **bottom center** of the pages starting from (1).

4. The page where chapter number and chapter name are specified should NOT be numbered.

Normal Body Text:

Font Size: 12 , Times New Roman, 1.5 Line spacing , single side writing

Paragraph Heading: Font Size: 14, Times New Roman, Bold

Source Code:

Font size:10, Times New Roman, 1 Line spacing

Important Note :

1. **Color:** **The color of the spiral bound file should be Blue for CSE with a transparent sheet on the front page.**
2. **There should be one for the Institute and one copy for each student.**
3. **The size of the appendix should not be more than one fourth of the total size of thesis.**
4. **The references should be sorted alphabetically author wise.**
5. **Presentation must be according to rules, mentioned on the last page of this document.**

References:

Atleast 10 research papers and 10 books should be included. The references should be referred to as [1],[2],[3] and in order in the thesis.

References :(Example)

[1] D.L. Carney, J.I. Cochran, "The 5ESS Switching System: Architectural Overview," *AT&T Technical Journal*, vol. **64,** no. **6,** July-August 1985, pp. 1339-1356.

[2] A. Stevens, *C++ Database Development*, MIS Press, New York, 1992, p.p. 34.

[3] J. Martin, *Computer Database Organization*, Prentice-Hall, Englewood Cliffs, NJ, 1977, p. 53.

INDUSTRIAL TRAINING

Submitted in partial fulfillment of the
Requirements for the award of the degree

of

Bachelor of Technology

in

Computer Science & Engineering

By:

Manjot Singh Dhillon (61/CSE2/2019)

Department of Computer Science & Engineering
Guru Tegh Bahadur Institute of Technology

Guru Gobind Singh Indraprastha University
Dwarka, New Delhi
Year 2017-2021

Auto-sorter Using PYTHON

Duration

01st June, 2019 – 30th July,2019

By:

Manjot Singh Dhillon (61/CSE2/2017)

At

Analytical Edupoint

Saket

New Delhi

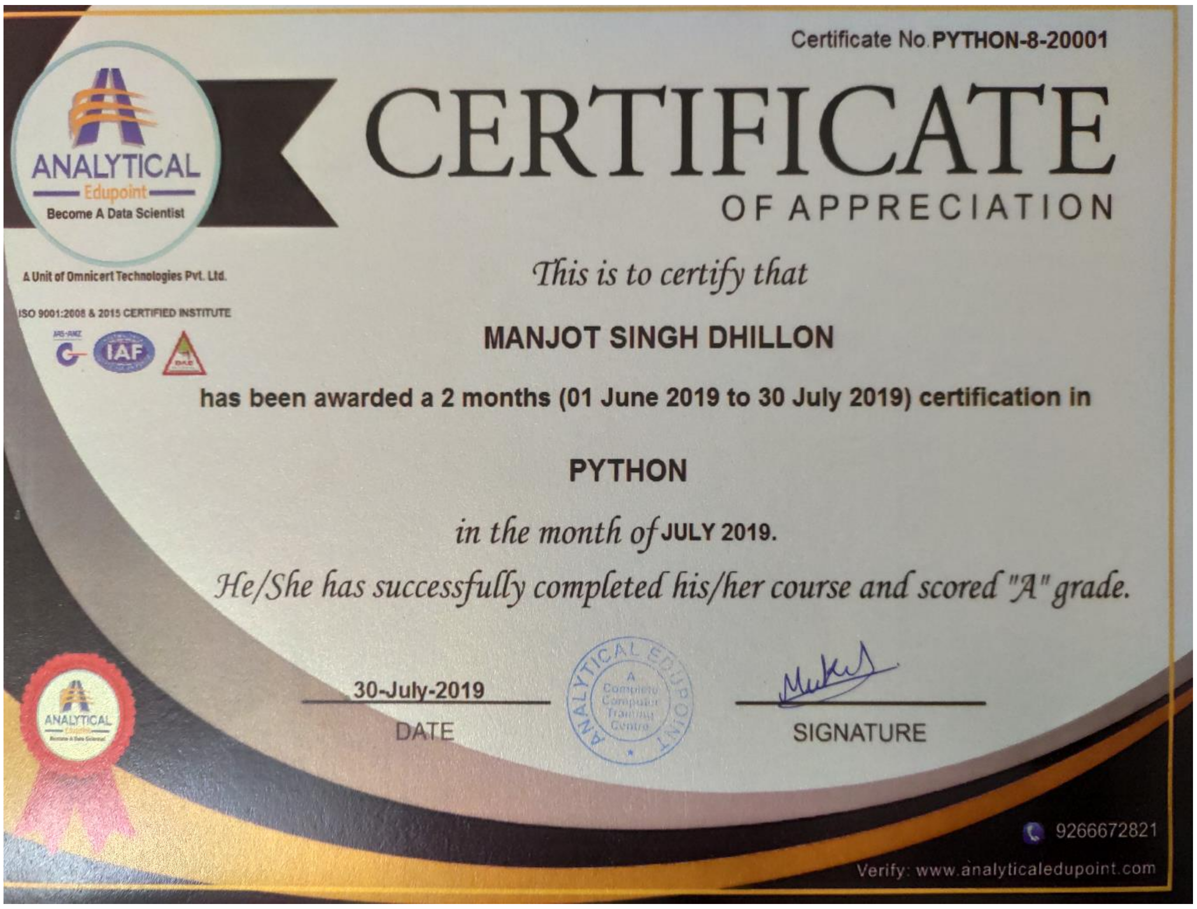
DECLARATION

I hereby declare that all the work presented in this Industrial Training Report for the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in **Computer Science & Engineering,** Guru Tegh Bahadur Institute of Technology, affiliated to Guru Gobind Singh Indraprastha University Delhi is an authentic record of our own work carried out at Analytical Edupoint from 1 June, 2019 to 30th July, 2019.

Date: 13th NOV, 2019.

Manjot Singh Dhillon (61/CSE2/2017)

CERTIFICATE



Certificate No. PYTHON-8-20001

ANALYTICAL Edupoint
Become A Data Scientist

A Unit of Omnicert Technologies Pvt. Ltd.

ISO 9001:2008 & 2015 CERTIFIED INSTITUTE

CERTIFICATE
OF APPRECIATION

This is to certify that

MANJOT SINGH DHILLON

has been awarded a 2 months (01 June 2019 to 30 July 2019) certification in

PYTHON

in the month of JULY 2019.

He/She has successfully completed his/her course and scored "A" grade.

30-July-2019
DATE

ANALYTICAL EDUPOINT
A Complete Computer Training Centre

[illegible]
SIGNATURE

9266672821

Verify: www.analyticaledupoint.com

Analytical Edupoint

Saket

The analytical EduPoint key aim is to deliver Job Oriented Training for both IT students and Non-IT students to perform excellence in their choice of domains or as per latest corporate requirements. Once candidates are given a thorough understanding of theory and practical sessions, they are asked to perform on live projects under the supervision of our experts.

F-76, Ground Floor
Near Saket Metro Station, Gate No-2 Saket, New Delhi - 110030
PH NO: +91-9266672821; 011-41620203
analyticaledupoint@gmail.com info@analyticaledupoint.com

ACKNOWLEDGEMENT

I would like to express my great gratitude towards **Mr.Mukul Das Gupta** who has given support and suggestions. Without his help i could not have presented this work upto the present standard. I also take this opportunity to give thanks to all others who gave me support for the project or in other aspects of my study at Guru Tegh Bahadur Institute of Technology.

Date : 13th NOV, 2019

Manjot Singh Dhillon (61/CSE2/2017)

9654293710ak@gmail.com

ABSTRACT

In modern day lives we do need a simplification in our lives.So that we can live stressfuly and freely.thou, once job life can be tough and challenging so by keeping that fact carefully in mind I made this project so that one can live happily.

This project is totally based on documents one get in his/her job.So when these documents got piled up in a folder in your own computer then he/she may feel difficulty working with documents and now I am not talking about only one type of document I am talking about all sort of documents,PDF’s,MP4’s,MP3’s,.png,.jpg,.ipynb and *e.t.c* .So basicaaly,this project do an automatically sort all those files in a specific way that all these files which are mismatched before would be arranged in specifics folder under their types.

Overall,a pdf file would be in the folder with all pdf’s files and similarly with all those files metion previously and unknown files would be under unknown section of folder.You just have to set the path of Source Folder and the Destination Folder then all files would be sorted under a minute.

This whole project is done using Python and performed using Jupyter notebook.All the imports are explained detailed later on this report.After numerous efforts I have got a fully-functional autosorter.

The beauty of this project is that it uses a simple for loop and if , else , elif conditions and basic imports which provide a great structure to program. The source code is also present in this report So,this project can also be understand by beginners and this code can run by a non coder easily.

REFERENCES

(1) Guttag, John V. (12 August 2016). Introduction to Computation and Programming Using Python: With Application to Understanding Data. MIT Press. ISBN 978-0-262-52962-4.
"Python 3.8.0".
(2)Peterson, Benjamin (19 October 2019). "Python 2.7.17 released". Python Insider. The Python Core Developers. Retrieved 22 October 2019.
(3)"PEP 483 -- The Theory of Type Hints". Python.org.
(4)File extension .pyo was removed in Python 3.5. See PEP 0488
(5)Holth, Moore (30 March 2014). "PEP 0441 -- Improving Python ZIP Application Support". Retrieved 12 November 2015.
(6)"Starlark Language". Retrieved 25 May 2019.
(7)"Why was Python created in the first place?". General Python FAQ. Python Software Foundation. Retrieved 22 March 2007.
(8)Kuchling, Andrew M. (22 December 2006). "Interview with Guido van Rossum (July 1998)". amk.ca. Archived from the original on 1 May 2007. Retrieved 12 March 2012.
(9)"itertools — Functions creating iterators for efficient looping — Python 3.7.1 documentation". docs.python.org.
(10)van Rossum, Guido (1993). "An Introduction to Python for UNIX/C Programmers". Proceedings of the NLUUG Najaarsconferentie (Dutch UNIX Users Group). CiteSeerX 10.1.1.38.2023. even though the design of C is far from ideal, its influence on Python is considerable.
(11)"Classes". The Python Tutorial. Python Software Foundation. Retrieved 20 February 2012. It is a mixture of the class mechanisms found in C++ and Modula-3
(12)Lundh, Fredrik. "Call By Object". effbot.org. Retrieved 21 November 2017. replace "CLU" with "Python", "record" with "instance", and "procedure" with "function or method", and you get a pretty accurate description of Python's object model.
(13)Simionato, Michele. "The Python 2.3 Method Resolution Order". Python Software Foundation. The C3 method itself has nothing to do with Python, since it was invented by people working on Dylan and it is described in a paper intended for lispers
(14)Kuchling, A. M. "Functional Programming HOWTO". Python v2.7.2 documentation. Python Software Foundation. Retrieved 9 February 2012.
(15)Schemenauer, Neil; Peters, Tim; Hetland, Magnus Lie (18 May 2001). "PEP 255 – Simple
(16)Generators". Python Enhancement Proposals. Python Software Foundation. Retrieved 9 February 2012.
(17)Smith, Kevin D.; Jewett, Jim J.; Montanaro, Skip; Baxter, Anthony (2 September 2004). "PEP 318 – Decorators for Functions and Methods". Python Enhancement Proposals. Python Software Foundation. Retrieved 24 February 2012.
(18) "More Control Flow Tools". Python 3 documentation. Python Software Foundation. Retrieved 24 July 2015.
(19)"CoffeeScript borrows chained comparisons from Python".
(20)"Genie Language - A brief guide". Retrieved 28 December 2015.
(21)"Perl and Python influences in JavaScript". www.2ality.com. 24 February 2013. Retrieved 15 May 2015.

CONTENTS

Chapter **Page No.**

Title Page i

Declaration and Certificate ii

Acknowledgement iii

Abstract iv

Tables and figures v

1. **Introduction** 1.

2.

2. Requirement Analysis with SRS

2.1

2.2

2.2.1

2.2.2

4.

5.

6.

7.

8.

9.

10 **Results and observations**

11 **Conclusions & future scope**

References

Appendix

TABLE OF CONTENTS

Title Page i

Declaration and Certificate ii

Acknowledgement iii

Abstract iv

Tables and figures v

INTRODUCTION **1**

PROBLEM DEFINITION 1

PROBLEM SOLUTION 3

REQUIREMENT ANALYIS WITH SRS **4**

WHAT IS SRS 4

WHY USE SRS 4

HOW TO WRITE SRS 5

REQUIREMENTS 6

THE JUPYTER NOTEBOOK .... **#**

ANACONDA .... **#**

PYTHON .... **#**

IMPORTS .... **#**

IMPORT OS .... **#**

IMPORT RE .... **#**

IMPORT TIME .... #

IMPORT SHUTIL .... **#**

CONCLUSION AND FUTURE SCOPE .... **#**

REFERENCES .... ***#***

APPENDIX .... ***#***

Chapter 1

INTRODUCTION

A document is a written, drawn, presented, or memorialized representation of thought. The word originates from the Latin documentum, which denotes a "teaching" or "lesson": the verb doceō denotes "to teach". In the past, the word was usually used to denote a written proof useful as evidence of a truth or fact. In the computer age, "document" usually denotes a primarily textual computer file, including its structure and format, e.g. fonts, colors, and images. Contemporarily, "document" is not defined by its transmission medium, e.g., paper, given the existence of electronic documents.

"Documentation" is distinct because it has more denotations than "document". Documents are also distinguished from "realia", which are three-dimensional objects that would otherwise satisfy the definition of "document" because they memorialize or represent thought; documents are considered more as 2 dimensional representations.

While documents are able to have large varieties of customization, all documents are able to be shared freely, and have the right to do so, creativity can be represented by documents, also. History, events, examples, opinion, etc. all can be expressed in documents.

One of the most popular and common document formats is the native format of Microsoft Word. Documents written and created in Word are typically saved as either DOC or DOCX depending on the Microsoft Office version you are using.

Problem Definition

These documents are to be considered in this project:

DOC and DOCX files can contain text and rich formatting as well as tables, images, and other objects. This makes them a preferable file format for different purposes that require a flexible way of formatting a document. From invoices or contracts with specific fields, to reports and essays containing graphs and tables, DOC and DOCX files provide a great way to express thoughts, present findings, and provide information.

Both files are able to be edited further, either by the original author or by the receiver. Furthermore, many programs are able to open DOC and DOCX files depending on their version. However, in some programs the formatting may change or images and tables may be displayed differently.

TXT

The best known and most compatible document type is the TXT document. TXT files contain raw text and can be created with any if not all word processing programs.

TXT files do neither contain formatting nor images or other objects. They are basic documents, developed to store text only. This makes them perfect for data storage or quick and easy information exchange—especially across platforms.

These raw text files can be opened by almost all programs like Microsoft Word, Pages for Mac, LibreOffice LaTex, and more. Of course, TXT files can be further edited as well.

PDF

PDF files were developed by Adobe and differ greatly from the document files mentioned above in several aspects. Next to formatted text, PDFs can contain images, graphs, tables, and even 3D drawings. Sometimes, they consist of images and scanned pages only instead of an underlying text file. Mixtures of both are possible as well.

PDF formats are perfect for printing since they preserve the formatting of the file they were created out of. Similar to an image, the PDF contains a fixed layout and always looks the same, no matter which program, software or device is used to open it. Next to the cost-free Adobe Reader, many programs are able to open PDF files. There are apps for mobile devices, and most browsers support opening PDF files as well. Furthermore, most word processing programs are able to export documents in the PDF format.

Unless using a program that specifically allows the modification of PDF files, they are protected from further editing. Furthermore, PDF files can be secured using a password or even encrypted.

The common distribution as well as the fact that everyone will see a PDF in the same way make it a perfect file format for spreading, sending and sharing documents.

HTM and HTML

HTM and HTML are abbreviations most commonly associated with web addresses. However, documents can also be saved in this format. Most word processing programs are able to save formatted text, containing images, audio and other objects in the HTML or HTM format. They can be opened with any web browser and are handled like real or true websites, completely with embedded media and outgoing links.

PPT and PPTX

Contrary to the file formats mentioned above, PPT and PPTX are known as presentation files rather than documents. They are native to the Microsoft PowerPoint program and can include images as well as text, graphs, animations, tables, slide transitions and embedded videos and audio files. It is the most common file used for different kinds of presentations among smaller or bigger groups of people.

The slideshows created using PowerPoint can be opened by other, open source programs as well. However, slide transitions, animations, and the overall formatting as well as the overall display may differ form the original one when PPT or PPTX files are opened in programs like Libre Office.

This whole project is done using Python and performed using Jupyter notebook.All the imports are explained detailed later on this report.After numerous efforts I have got a fully-functional autosorter.

Problem Solution

The beauty of this project is that it uses a simple for loop and if , else , elif conditions and basic imports which provide a great structure to program. The source code is also present in this report So,this project can also be understand by beginners and this code can run by a non coder easily.

To simplify this project the Anaconda platform is used in which Jupyter notebook is used to perform the python coding containing simple for loops and if else condition and basic imports of Time,OS,Shutil,RE.

The autosorter can be used by everyone who need to sort files and documents. Now we will learn everything about this project in detail in later sections.

Chapter 2

Requirement Analysis with SRS

What Is in a Software Requirements Specification Document?

A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform.

An SRS describes the functionality the product needs to fulfill all stakeholders (business, users) needs.

A typical SRS includes:

A purpose
An overall description
Specific requirements
The best SRS documents define how the software will interact when embedded in hardware — or when connected to other software. Good SRS documents also account for real-life users.

Why Use an SRS Document?

A software requirements specification is the basis for your entire project. It lays the framework that every team involved in development will follow.

It's used to provide critical information to multiple teams — development, quality assurance, operations, and maintenance. This keeps everyone on the same page.

Using the SRS helps to ensure requirements are fulfilled. And it can also help you make decisions about your product's lifecycle — for instance, when to retire a feature.

Writing an SRS can also minimize overall development time and costs. Embedded development teams especially benefit from using an SRS.

Software Requirements Specification vs. System Requirements Specification
A software requirements specification (SRS) includes in-depth descriptions of the software that will be developed.

A system requirements specification (SyRS) collects information on the requirements for a system.

"Software" and "system" are sometimes used interchangeably as SRS. But, a software requirement specification provides greater detail than a system requirements specification.

How to Write an SRS Document?

Writing an SRS document is important. But it isn't always easy to do.

Here are five steps you can follow to write an effective SRS document.

1. Create an Outline (Or Use an SRS Template)
Your first step is to create an outline for your software requirements specification. This may be something you create yourself. Or you may use an existing SRS template.

If you're creating this yourself, here's what your outline might look like:

1. Introduction

1.1 Purpose

1.2 Intended Audience

1.3 Intended Use

1.4 Scope

1.5 Definitions and Acronyms

2. Overall Description

2.1 User Needs

2.2 Assumptions and Dependencies

3. System Features and Requirements

    3.1 Functional Requirements

    3.2 External Interface Requirements

    3.3 System Features

    3.4 Nonfunctional Requirements

Once you have your basic outline, you're ready to start filling it out.

2. Start With a Purpose
The introduction to your SRS is very important. It sets the expectation for the product you're building.

So, start by defining the purpose of your product.

Intended Audience and Intended Use
Define who in your organization will have access to the SRS — and how they should use it. This may include developers, testers, and project managers. It could also include stakeholders in other departments, including leadership teams, sales, and marketing.

Product Scope
Describe the software being specified. And include benefits, objectives, and goals. This should relate to overall business goals, especially if teams outside of development will have

access to the SRS.

Definitions and Acronyms
It’s smart to include a risk definition. Avoiding risk is top-of-mind for many developers — especially those working on safety-critical development teams.

Here’s an example. If you’re creating a medical device, the risk might be the device fails and causes a fatality.

By defining that risk up front, it’s easier to determine the specific requirements you’ll need to mitigate it.

3. Give an Overview of What You’ll Build
Your next step is to give a description of what you’re going to build. Is it an update to an existing product? Is it a new product? Is it an add-on to a product you’ve already created?

These are important to describe upfront, so everyone knows what you’re building.

You should also describe why you’re building it and who it’s for.

User Needs
User needs — or user classes and characteristics — are critical. You’ll need to define who is going to use the product and how.

You’ll have primary and secondary users who will use the product on a regular basis. You may also need to define the needs of a separate buyer of the product (who may not be a primary/secondary user). And, for example, if you’re building a medical device, you’ll need to describe the patient’s needs.

Assumptions and Dependencies
There might be factors that impact your ability to fulfill the requirements outlined in your SRS. What are those factors?

Are there any assumptions you’re making with the SRS that could turn out to be false? You should include those here, as well.

Finally, you should note if your project is dependent on any external factors. This might include software components you’re reusing from another project.

4. Detail Your Specific Requirements
The next section is key for your development team. This is where you detail the specific requirements for building your product.

Requirements

Functional requirements are essential to building your product.

If you’re developing a medical device, these requirements may include infusion and battery. And within these functional requirements, you may have a subset of risks and requirements.

External Interface Requirements
External interface requirements are types of functional requirements. They’re important for embedded systems. And they outline how your product will interface with other components.

There are several types of interfaces you may have requirements for, including:

User
Hardware
Software
Communications
System Features
System features are types of functional requirements. These are features that are required in order for a system to function.

Other Nonfunctional Requirements
Nonfunctional requirements can be just as important as functional ones.

These include:

Performance
Safety
Security
Quality
The importance of this type of requirement may vary depending on your industry. Safety requirements, for example, will be critical in the medical device industry.

IEEE also provides guidance for writing software requirements specifications, if you're a member.

5. Get Approval for the SRS
Once you've completed the SRS, you'll need to get it approved by key stakeholders. And everyone should be reviewing the latest version of the document.

Chapter 3

The Jupyter Notebook

Jupyter Notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a JSON document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdown), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) through "Download As" in the web interface, via the nbconvert library or "jupyter nbconvert" command line interface in a shell.

To simplify visualisation of Jupyter notebook documents on the web, the nbconvert library is provided as a service through NbViewer which can take a URL to any publicly available notebook document, convert it to HTML on the fly and display it to the user.

Jupyter Notebook provides a browser-based REPL built upon a number of popular open-source libraries:

•IPython

•ØMQ

•Tornado (web server)

•jQuery

•Bootstrap (front-end framework)

•MathJax

Jupyter Notebook can connect to many kernels to allow programming in many languages. By default Jupyter Notebook ships with the IPython kernel. As of the 2.3 release[8][9] (October 2014), there are currently 49 Jupyter-compatible kernels for many programming languages, including Python, R, Julia and Haskell.[10]

The Notebook interface was added to IPython in the 0.12 release[11] (December 2011), renamed to Jupyter notebook in 2015 (IPython 4.0 – Jupyter 1.0). Jupyter Notebook is similar to the notebook interface of other programs such as Maple, Mathematica, and SageMath, a computational interface style that originated with Mathematica in the 1980s[12]. According to The Atlantic, Jupyter interest overtook the popularity of the Mathematica notebook interface in early 2018[12].

Jupyter kernels

A Jupyter kernel is a program responsible for handling various types of request (code execution, code completions, inspection), and providing a reply. Kernels talk to the other components of Jupyter using ZeroMQ over the network, and thus can be on the same or remote machines.

Unlike many other Notebook-like interfaces, in Jupyter, kernels are not aware that they are attached to a specific document, and can be connected to many clients at once. Usually kernels allow execution of only a single language, but there are a couple of exceptions.

By default Jupyter ships with IPython as a default kernel and a reference implementation via the ipykernel wrapper. Kernels for many languages having varying quality and features are available.

JupyterHub

JupyterHub[13] is a multi-user server for Jupyter Notebooks. It is designed to support many users by spawning, managing, and proxying many singular Jupyter Notebook servers.[citation needed] While JupyterHub requires managing servers, third-party services like Jupyo[14] provide an alternative to JupyterHub by hosting and managing multi-user Jupyter notebooks in the cloud.

JupyterLab

JupyterLab is the next-generation user interface for Project Jupyter. It offers all the familiar building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a flexible and powerful user interface. The first stable release was announced on February 20, 2018.

Chapter 4

Anaconda

Anaconda distribution comes with more than 1,500 packages as well as the conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator[7], as a graphical alternative to the command line interface (CLI).
The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.
When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example, Google Tensorflow, can find that it stops working having used pip to install a different package that requires a different version of the dependent numpy library than the one uses by Tensorflow. In some cases, the package may appear to work but produce different results in detail.
In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g. the user may wish to have Tensorflow version 2,0 or higher), works out how to install a compatible set of dependencies, warning if this cannot be done.
Open source packages can be individually installed from the Anaconda repository[8], Anaconda Cloud (anaconda.org), or your own private repository or mirror, using the conda install command. Anaconda Inc compiles and builds all the packages in the Anaconda repository itself, and provides binaries for Windows 32/64 bit, Linux 64 bit and MacOS 64-bit. Anything available on PyPI may be installed into a conda environment using pip, and conda will keep track of what it has installed itself and what pip has installed.
Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud,[9] PyPI or other repositories.
The default installation of Anaconda2 includes Python 2.7 and Anaconda3 includes Python 3.7. However, it is possible to create new environments that include any version of Python packaged with conda[10].
Anaconda Navigator
Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows users to launch applications and manage conda packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository, install them in an environment, run the packages and update them. It is available for Windows, macOS and Linux.
The following applications are available by default in Navigator[11]:

- JupyterLab
- Jupyter Notebook
- QtConsole
- Spyder
- Glueviz
- Orange
- Rstudio
- Visual Studio Code

Conda
Conda is an open source,[12] cross-platform,[13] language-agnostic[14] package manager and environment management system[15][16][17] that installs, runs, and updates packages and their dependencies.[12] It was created for Python programs, but it can package and distribute software for any language (e.g., R), including multi-language projects.[14] The conda package and environment manager is included in all versions of Anaconda, Miniconda,[18] and Anaconda Repository.[8]
Anaconda Cloud

Anaconda Cloud is a package management service by Anaconda where you can find, access, store and share public and private notebooks, environments, and conda and PyPI packages. Cloud hosts useful Python packages, notebooks and environments for a wide variety of applications. You do not need to log in or to have a Cloud account, to search for public packages, download and install them.
You can build new packages using the Anaconda Client command line interface (CLI), then manually or automatically upload t

Chapter 5

Python

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.[27]
Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.[28]
Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3.
The Python 2 language, i.e. Python 2.7.x, is "sunsetting" on January 1, 2020 (after extension; first planned for 2015), and the Python team of volunteers will not fix security issues, or improve it in other ways after that date.[29][30] With the end-of-life, only Python 3.5.x and later will be supported.
Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, an open source[31] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.
Python was conceived in the late 1980s[32] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL),[33] capable of exception handling and interfacing with the Amoeba operating system.[8] Its implementation began in December 1989.[34] Van Rossum shouldered sole responsibility for the project, as the lead developer, until July 12, 2018, when he announced his "permanent vacation" from his responsibilities as Python's Benevolent Dictator For Life, a title the Python community bestowed upon him to reflect his long-term commitment as the project's chief decision-maker.[35] He now shares his leadership as a member of a five-person steering council.[36][37][38] In January, 2019, active Python core developers elected Brett Cannon, Nick Coghlan, Barry Warsaw, Carol Willing and Van Rossum to a five-member "Steering Council" to lead the project.[39]

Python 2.0 was released on 16 October 2000 with many major new features, including a cycle-detecting garbage collector and support for Unicode.[40]

Python 3.0 was released on 3 December 2008. It was a major revision of the language that is not completely backward-compatible.[41] Many of its major features were backported to Python 2.6.x[42] and 2.7.x version series. Releases of Python 3 include the 2to3 utility, which automates (at least partially) the translation of Python 2 code to Python 3.[43]

Python 2.7's end-of-life date was initially set at 2015 then postponed to 2020 out of concern that a large body of existing code could not easily be forward-ported to Python 3.[44][45]

Features and philosophy
Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional

programming and aspect-oriented programming (including by metaprogramming[46] and metaobjects (magic methods)).[47] Many other paradigms are supported via extensions, including design by contract[48][49] and logic programming.[50]

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution.

Python's design offers some support for functional programming in the Lisp tradition. It has filter, map, and reduce functions; list comprehensions, dictionaries, sets, and generator expressions.[51] The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.[52]

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:[53]

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Readability counts.
Rather than having all of its functionality built into its core, Python was designed to be highly extensible. This compact modularity has made it particularly popular as a means of adding programmable interfaces to existing applications. Van Rossum's vision of a small core language with a large standard library and easily extensible interpreter stemmed from his frustrations with ABC, which espoused the opposite approach.[32]

Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology. In contrast to Perl's "there is more than one way to do it" motto, Python embraces a "there should be one—and preferably only one—obvious way to do it" design philosophy.[53] Alex Martelli, a Fellow at the Python Software Foundation and Python book author, writes that "To describe something as 'clever' is not considered a compliment in the Python culture."[54]

Python's developers strive to avoid premature optimization, and reject patches to non-critical parts of the CPython reference implementation that would offer marginal increases in speed at the cost of clarity.[55] When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or use PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter.

An important goal of Python's developers is keeping it fun to use. This is reflected in the language's name—a tribute to the British comedy group Monty Python[56]—and in occasionally playful approaches to tutorials and reference materials, such as examples that refer to spam and eggs (from a famous Monty Python sketch) instead of the standard foo and bar.[57][58]

A common neologism in the Python community is pythonic, which can have a wide range of meanings related to program style. To say that code is pythonic is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called unpythonic.
Users and admirers of Python, especially those considered knowledgeable or experienced,

are often referred to as Pythonistas.

Syntax and semantics

Main article: Python syntax and semantics

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.[61]

Indentation

Main article: Python syntax and semantics § Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.[62] Thus, the program's visual structure accurately represents the program's semantic structure.[1]

Statements and control flow

Python's statements include (among others):

The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound. Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing.
The if statement, which conditionally executes a block of code, along with else and elif (a contraction of else-if).
The for statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
The while statement, which executes a block of code as long as its condition is true.
The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
The raise statement, used to raise a specified exception or re-raise a caught exception.
The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
The def statement, which defines a function or method.
The with statement, from Python 2.5 released on September 2006,[63] which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behavior and replaces a common try/finally idiom.[64]
The pass statement, which serves as a NOP. It is syntactically needed to create an empty

code block.
The assert statement, used during debugging to check for conditions that ought to apply.
The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.
The import statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import * or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>], ...
The print statement was changed to the print() function in Python 3.[65]
Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will.[66][67] However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators.[68] Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.[69]

Expressions
Some Python expressions are similar to languages such as C and Java, while some are not:

Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating point/division.[70] Python also added the ** operator for exponentiation.
From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.[71][72]
From Python 3.8, the syntax :=, called as 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.[73]
In Python, == compares by value, versus Java, which compares numerics by value[74] and objects by reference.[75] (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c.
Python uses the words and, or, not for its boolean operators rather than the symbolic &&, ||, ! used in Java and C.
Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression.[51]
Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.
Conditional expressions in Python are written as x if c else y[76] (different in order of operands from the c ? x : y operator common to many other languages).
Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.[77]
Python features sequence unpacking where multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an iterable object on the right hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the

produced values to the corresponding expression on the left.[78]
Python has a "string format" operator %. This functions analogous to printf format strings in C, e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2". In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'.[79]
Python has various kinds of string literals:
Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".[79]
Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.
Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.
Python has array index and array slicing expressions on lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index. The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a shallow copy.
In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

List comprehensions vs. for-loops
Conditional expressions vs. if blocks
The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.
Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code but if c = 1: ... causes a syntax error in Python.

Methods
Methods on objects are functions attached to the object's class; the syntax instance.method(argument) is, for normal methods and functions, syntactic sugar for Class.method(instance, argument). Python methods have an explicit self parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).[80]

Chapter 6

Imports

The Python programming language comes with a variety of built-in functions. Among these are several common functions, including:

print() which prints expressions out
abs() which returns the absolute value of a number
int() which converts another data type to an integer
len() which returns the length of a sequence or collection
These built-in functions, however, are limited, and we can make use of modules to make more sophisticated programs.

Modules are Python .py files that consist of Python code. Any Python file can be referenced as a module. A Python file called hello.py has the module name of hello that can be imported into other Python files or used on the Python command line interpreter. You can learn about creating your own modules by reading How To Write Modules in Python 3.

Modules can define functions, classes, and variables that you can reference in other Python .py files or via the Python command line interpreter.

In Python, modules are accessed by using the import statement. When you do this, you execute the code of the module, keeping the scopes of the definitions so that your current file(s) can make use of these.

When Python imports a module called hello for example, the interpreter will first search for a built-in module called hello. If a built-in module is not found, the Python interpreter will then search for a file named hello.py in a list of directories that it receives from the sys.path variable.

This tutorial will walk you through checking for and installing modules, importing modules, and aliasing modules.

Checking For and Installing Modules
There are a number of modules that are built into the Python Standard Library, which contains many modules that provide access to system functionality or provide standardized solutions. The Python Standard Library is part of every Python installation.
Importing Modules
To make use of the functions in a module, you'll need to import the module with an import statement.

An import statement is made up of the import keyword along with the name of the module.

In a Python file, this will be declared at the top of the code, under any shebang lines or general comments.

So, in the Python program file my_rand_int.py we would import the random module to generate random numbers in this manner:

```
my_rand_int.py
import random
```

When we import a module, we are making it available to us in our current program as a separate namespace. This means that we will have to refer to the function in dot notation, as in [module].[function].

In practice, with the example of the random module, this may look like a function such as:

random.randint() which calls the function to return a random integer, or
random.randrange() which calls the function to return a random element from a specified range.
Let’s create a for loop to show how we will call a function of the random module within our my_rand_int.py program:

```
my_rand_int.py
import random


for i in range(10):
    print(random.randint(1, 25))
```

This small program first imports the random module on the first line, then moves into a for loop which will be working with 10 elements. Within the loop, the program will print a random integer within the range of 1 through 25 (inclusive). The integers 1 and 25 are passed to random.randint() as its parameters.

To make use of the functions in a module, you'll need to import the module with an import statement. An import statement is made up of the import keyword along with the name of the module. In a Python file, this will be declared at the top of the code, under any shebang lines or general comments

Chapter 7

Import OS

```
NAME
    os - OS routines for NT or Posix depending on what system we're on.

DESCRIPTION
    This exports:
      - all functions from posix or nt, e.g. unlink, stat, etc.
      - os.path is either posixpath or ntpath
      - os.name is either 'posix' or 'nt'
      - os.curdir is a string representing the current directory (always '.')
      - os.pardir is a string representing the parent directory (always '..')
      - os.sep is the (or a most common) pathname separator ('/' or '\\')
      - os.extsep is the extension separator (always '.')
      - os.altsep is the alternate pathname separator (None or '/')
      - os.pathsep is the component separator used in $PATH etc
      - os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
      - os.defpath is the default search path for executables
      - os.devnull is the file path of the null device ('/dev/null', etc.)

    Programs that import and use 'os' stand a better chance of being
    portable between different platforms.  Of course, they must then
    only use functions that are defined by all platforms (e.g., unlink
    and opendir), and leave all pathname manipulation to os.path
    (e.g., split and join).

CLASSES
    builtins.Exception(builtins.BaseException)
        builtins.OSError
    builtins.object
        nt.DirEntry
    builtins.tuple(builtins.object)
        nt.times_result
        nt.uname_result
        stat_result
        statvfs_result
        terminal_size

    class DirEntry(builtins.object)
     |  Methods defined here:
     |
     |  __fspath__(self, /)
     |      Returns the path for the entry.
     |
     |  __repr__(self, /)
     |      Return repr(self).
     |
     |  inode(self, /)
     |      Return inode of the entry; cached per entry.
     |
     |  is_dir(self, /, *, follow_symlinks=True)
     |      Return True if the entry is a directory; cached per entry.
```

```
 |
 |  is_file(self, /, *, follow_symlinks=True)
 |      Return True if the entry is a file; cached per entry.
 |
 |  is_symlink(self, /)
 |      Return True if the entry is a symbolic link; cached per entry.
 |
 |  stat(self, /, *, follow_symlinks=True)
 |      Return stat_result object for the entry; cached per entry.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  name
 |      the entry's base filename, relative to scandir() "path" argument
 |
 |  path
 |      the entry's full path name; equivalent to os.path.join(scandir_path, entry.name)
```

Chapter 8

Import RE

NAME
    re - Support for regular expressions (RE).

DESCRIPTION
    This module provides regular expression matching operations similar to those found in Perl. It supports both 8-bit and Unicode strings; both the pattern and the strings being processed can contain null bytes and characters outside the US ASCII range.

    Regular expressions can contain both special and ordinary characters. Most ordinary characters, like "A", "a", or "0", are the simplest regular expressions; they simply match themselves. You can concatenate ordinary characters, so last matches the string 'last'.

    The special characters are:

```
        "."      Matches any character except a newline.
        "^"      Matches the start of the string.
        "$"      Matches the end of the string or just before the newline at
                 the end of the string.
        "*"      Matches 0 or more (greedy) repetitions of the preceding RE.
                 Greedy means that it will match as many repetitions as possible.
        "+"      Matches 1 or more (greedy) repetitions of the preceding RE.
        "?"      Matches 0 or 1 (greedy) of the preceding RE.
        *?,+?,?? Non-greedy versions of the previous three special characters.
        {m,n}    Matches from m to n repetitions of the preceding RE.
        {m,n}?   Non-greedy version of the above.
        "\\"     Either escapes special characters or signals a special sequence.
        []       Indicates a set of characters.
                 A "^" as the first character indicates a complementing set.
        "|"      A|B, creates an RE that will match either A or B.
        (...)    Matches the RE inside the parentheses.
                 The contents can be retrieved or matched later in the string.
        (?aiLmsux) Set the A, I, L, M, S, U, or X flag for the RE (see below).
        (?:...)  Non-grouping version of regular parentheses.
        (?P<name>...) The substring matched by the group is accessible by name.
        (?P=name)     Matches the text matched earlier by the group named name.
        (?#...)  A comment; ignored.
        (?=...)  Matches if ... matches next, but doesn't consume the string.
        (?!...)  Matches if ... doesn't match next.
        (?<=...) Matches if preceded by ... (must be fixed length).
        (?<!...) Matches if not preceded by ... (must be fixed length).
        (?(id/name)yes|no) Matches yes pattern if the group with id/name matched,
                           the (optional) no pattern otherwise.
```

    The special sequences consist of "\\" and a character from the list below. If the ordinary character is not on the list, then the resulting RE will match the second character.

```
        \number  Matches the contents of the group of the same number.
        \A       Matches only at the start of the string.
        \Z       Matches only at the end of the string.
        \b       Matches the empty string, but only at the start or end of a word.
```

\B	Matches the empty string, but not at the start or end of a word.
\d	Matches any decimal digit; equivalent to the set [0-9] in bytes patterns or string patterns with the ASCII flag. In string patterns without the ASCII flag, it will match the whole range of Unicode digits.
\D	Matches any non-digit character; equivalent to [^\d].
\s	Matches any whitespace character; equivalent to [\t\n\r\f\v] in bytes patterns or string patterns with the ASCII flag. In string patterns without the ASCII flag, it will match the whole range of Unicode whitespace characters.
\S	Matches any non-whitespace character; equivalent to [^\s].
\w	Matches any alphanumeric character; equivalent to [a-zA-Z0-9_] in bytes patterns or string patterns with the ASCII flag. In string patterns without the ASCII flag, it will match the range of Unicode alphanumeric characters (letters plus digits plus underscore). With LOCALE, it will match the set [0-9_] plus characters defined as letters for the current locale.
\W	Matches the complement of \w.
\\	Matches a literal backslash.

This module exports the following functions:

match	Match a regular expression pattern to the beginning of a string.
fullmatch	Match a regular expression pattern to all of a string.
search	Search a string for the presence of a pattern.
sub	Substitute occurrences of a pattern found in a string.
subn	Same as sub, but also return the number of substitutions made.
split	Split a string by the occurrences of a pattern.
findall	Find all occurrences of a pattern in a string.
finditer	Return an iterator yielding a Match object for each match.

Chapter 9

Import Time

```
NAME
    time - This module provides various functions to manipulate time values.

DESCRIPTION
    There are two standard representations of time.  One is the number
    of seconds since the Epoch, in UTC (a.k.a. GMT).  It may be an integer
    or a floating point number (to represent fractions of seconds).
    The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
    The actual value can be retrieved by calling gmtime(0).

    The other representation is a tuple of 9 integers giving local time.
    The tuple items are:
      year (including century, e.g. 1998)
      month (1-12)
      day (1-31)
      hours (0-23)
      minutes (0-59)
      seconds (0-59)
      weekday (0-6, Monday is 0)
      Julian day (day in the year, 1-366)
      DST (Daylight Savings Time) flag (-1, 0 or 1)
    If the DST flag is 0, the time is given in the regular time zone;
    if it is 1, the time is given in the DST time zone;
    if it is -1, mktime() should guess based on the date and time.

CLASSES
    builtins.tuple(builtins.object)
        struct_time

    class struct_time(builtins.tuple)
     |  struct_time(iterable=(), /)
     |
     |  The time value as returned by gmtime(), localtime(), and strptime(), and
     |  accepted by asctime(), mktime() and strftime().  May be considered as a
     |  sequence of 9 integers.
     |
     |  Note that several fields' values are not the same as those defined by
     |  the C language standard for struct tm.  For example, the value of the
     |  field tm_year is the actual year, not year - 1900.  See individual
     |  fields' descriptions for details.
     |
     |  Method resolution order:
     |      struct_time
     |      builtins.tuple
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __reduce__(...)
     |      Helper for pickle.
     |
     |  __repr__(self, /)
```

```
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  tm_gmtoff
 |      offset from UTC in seconds
 |
 |  tm_hour
 |      hours, range [0, 23]
 |
 |  tm_isdst
 |      1 if summer time is in effect, 0 if not, and -1 if unknown
 |
 |  tm_mday
 |      day of month, range [1, 31]
 |
 |  tm_min
 |      minutes, range [0, 59]
 |
 |  tm_mon
 |      month of year, range [1, 12]
 |
 |  tm_sec
 |      seconds, range [0, 61])
 |
 |  tm_wday
 |      day of week, range [0, 6], Monday is 0
 |
 |  tm_yday
 |      day of year, range [1, 366]
 |
 |  tm_year
 |      year, for example, 1993
 |
 |  tm_zone
 |      abbreviation of timezone name
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  n_fields = 11
 |
 |  n_sequence_fields = 9
 |
 |  n_unnamed_fields = 0
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.tuple:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present
```

Chapter 10

Import Shutil

```
NAME
    shutil - Utility functions for copying and archiving files and directory trees.

DESCRIPTION
    XXX The functions here don't copy the resource fork or other metadata on Mac.

CLASSES
    builtins.OSError(builtins.Exception)
        Error
            SameFileError
        ExecError
        SpecialFileError

    class Error(builtins.OSError)
     |  Base class for I/O related errors.
     |
     |  Method resolution order:
     |      Error
     |      builtins.OSError
     |      builtins.Exception
     |      builtins.BaseException
     |      builtins.object
     |
     |  Data descriptors defined here:
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from builtins.OSError:
     |
     |  __init__(self, /, *args, **kwargs)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  __reduce__(...)
     |      Helper for pickle.
     |
     |  __str__(self, /)
     |      Return str(self).
     |
     |  ----------------------------------------------------------------------
     |  Static methods inherited from builtins.OSError:
     |
     |  __new__(*args, **kwargs) from builtins.type
     |      Create and return a new object.  See help(type) for accurate signature.
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors inherited from builtins.OSError:
     |
|  characters_written
|
|  errno
|      POSIX exception code
|
|  filename
|      exception filename
|
|  filename2
|      second exception filename
|
|  strerror
|      exception strerror
|
|  winerror
|      Win32 exception code
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.BaseException:
|
|  __delattr__(self, name, /)
|      Implement delattr(self, name).
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __setattr__(self, name, value, /)
|      Implement setattr(self, name, value).
|
|  __setstate__(...)
|
|  with_traceback(...)
|      Exception.with_traceback(tb) --
|      set self.__traceback__ to tb and return self.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from builtins.BaseException:
|
|  __cause__
|      exception cause
|
|  __context__
|      exception context
|
|  __dict__
|
|  __suppress_context__
|
|  __traceback__
|
|  args
```

```
class ExecError(builtins.OSError)
 |  Raised when a command could not be executed
 |
 |  Method resolution order:
 |      ExecError
 |      builtins.OSError
 |      builtins.Exception
 |      builtins.BaseException
 |      builtins.object
 |
 |  Data descriptors defined here:
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.OSError:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from builtins.OSError:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from builtins.OSError:
 |
 |  characters_written
 |
 |  errno
 |      POSIX exception code
 |
 |  filename
 |      exception filename
 |
 |  filename2
 |      second exception filename
 |
 |  strerror
 |      exception strerror
 |
 |  winerror
 |      Win32 exception code
 |
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.BaseException:
|
|  __delattr__(self, name, /)
|      Implement delattr(self, name).
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __repr__(self, /)
|      Return repr(self).
|
|  __setattr__(self, name, value, /)
|      Implement setattr(self, name, value).
|
|  __setstate__(...)
|
|  with_traceback(...)
|      Exception.with_traceback(tb) --
|      set self.__traceback__ to tb and return self..
```

Chapter 11

CONCLUSION AND FUTURE SCOPE

CONCLUSION

Working

Overall,a pdf file would be in the folder with all pdf's files and similarly with all those files metion previously and unknown files would be under unknown section of folder.You just have to set the path of Source Folder and the Destination Folder then all files would be sorted under a minute.

This whole project is done using Python and performed using Jupyter notebook.All the imports are explained detailed later on this report.After numerous efforts I have got a fully-functional autosorter.

The beauty of this project is that it uses a simple for loop and if , else , elif conditions and basic imports which provide a great structure to program. The source code is also present in this report So,this project can also be understand by beginners and this code can run by a non coder easily.

Goals

This project is totally based on documents one get in his/her job.So when these documents got piled up in a folder in your own computer then he/she may feel difficulty working with documents and now I am not talking about only one type of document I am talking about all sort of documents,PDF's,MP4's,MP3's,.png,.jpg,.ipynb and *e.t.c* .So basicaaly,this project do an automatically sort all those files in a specific way that all these files which are mismatched before would be arranged in specifics folder under their types.

Result

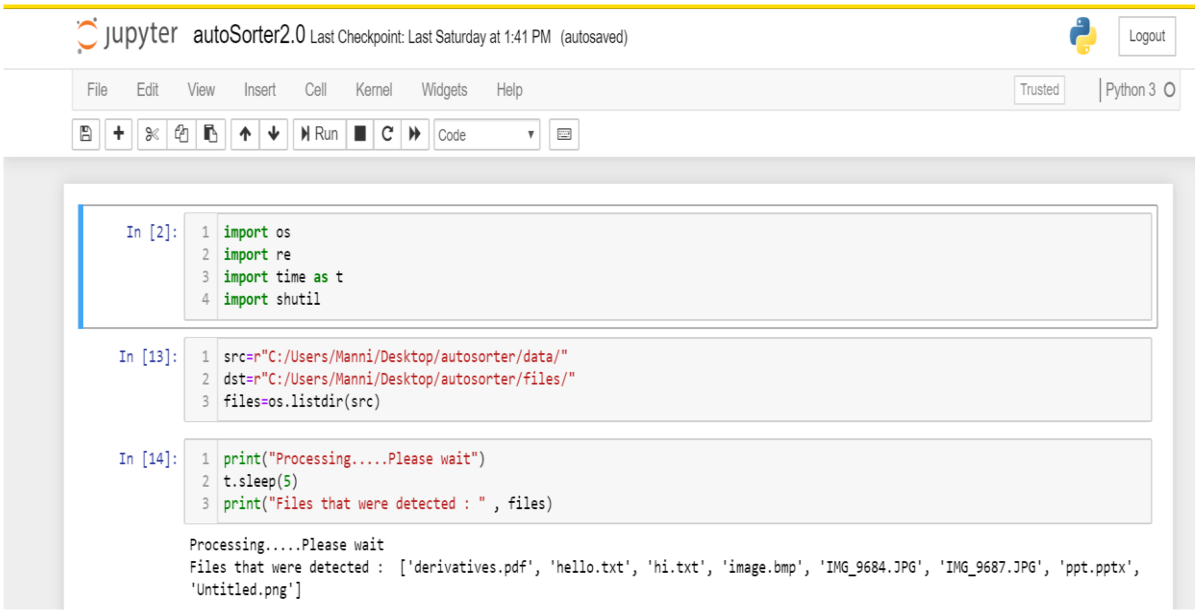
Favourable outcome have been achieved all files are sorted and can be located at the destination folder.{files}

Future Scope

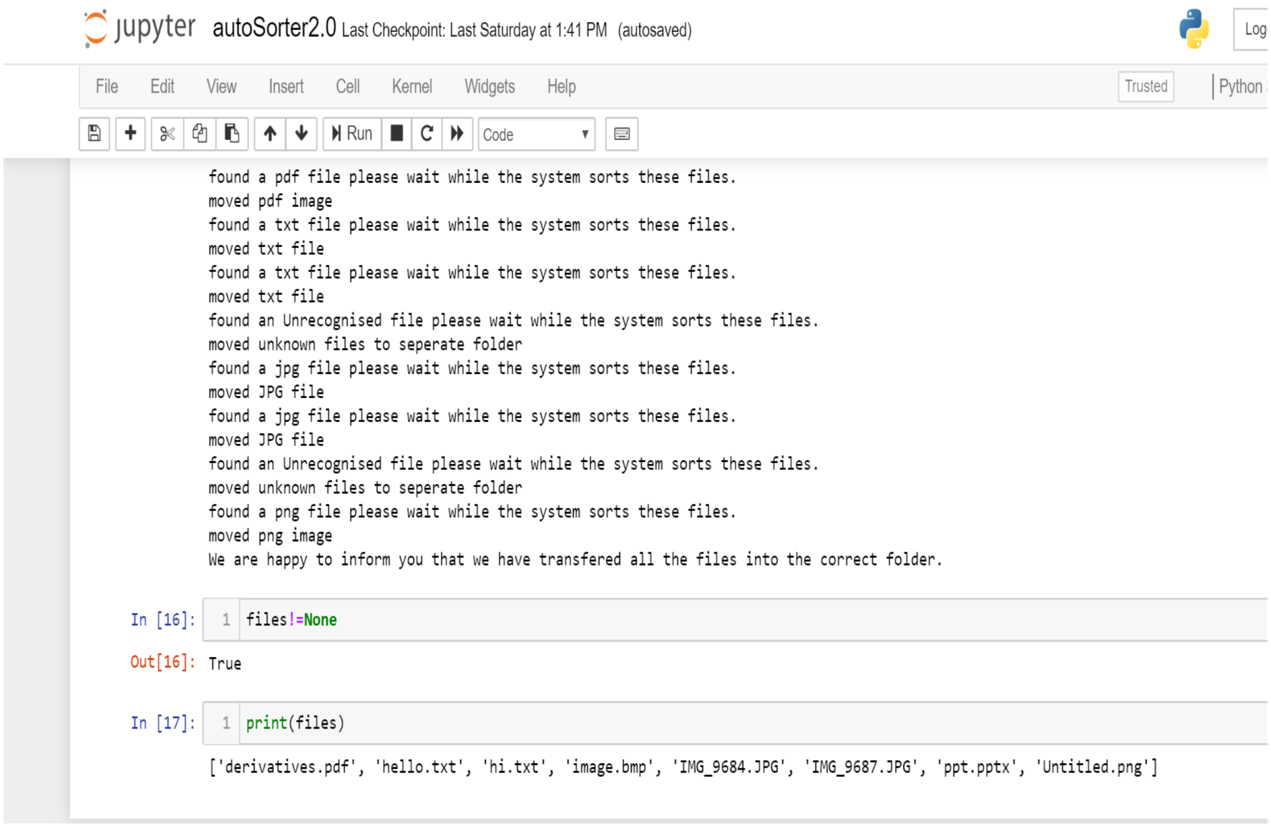
Any software have their own drawbacks thou they have a wear and tear properties in upcoming future and this project may be come down due to latest technology or a more smart program which can do the same task in efficient time.But for now this program holds it position and is reliable and efficient.

APPENDIX A

OUTPUT
BEST CASE:

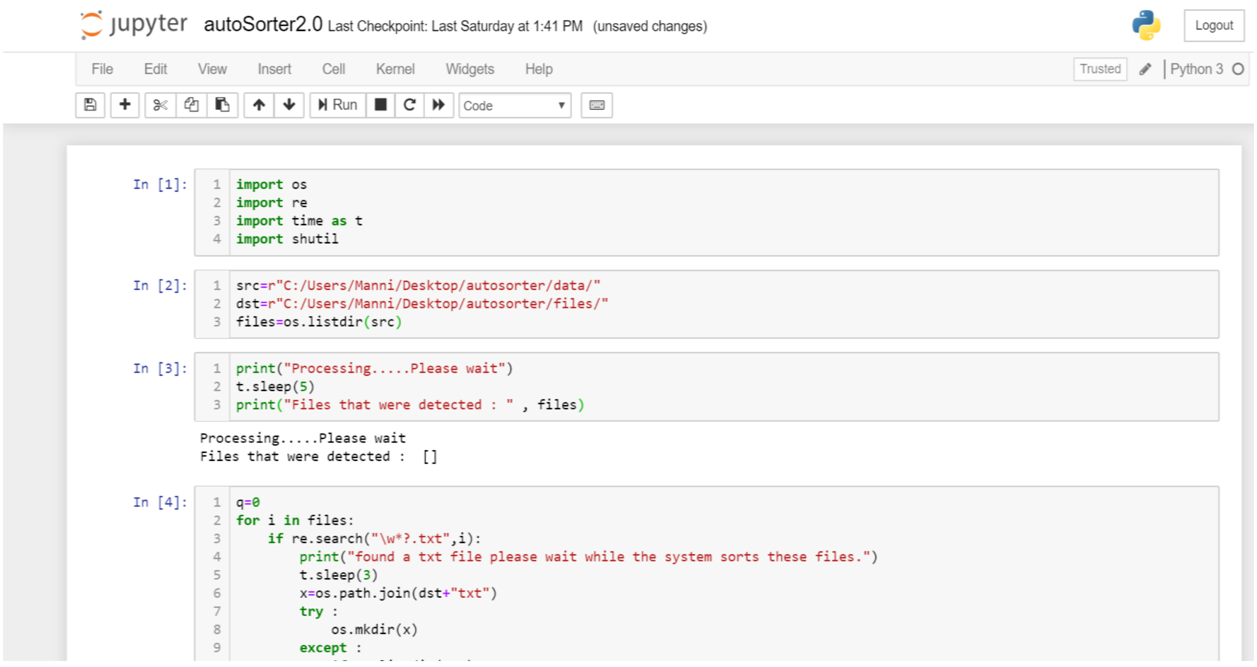

jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [2]:  import os
         import re
         import time as t
         import shutil

In [13]: src=r"C:/Users/Manni/Desktop/autosorter/data/"
         dst=r"C:/Users/Manni/Desktop/autosorter/files/"
         files=os.listdir(src)

In [14]: print("Processing.....Please wait")
         t.sleep(5)
         print("Files that were detected : " , files)

         Processing.....Please wait
         Files that were detected :  ['derivatives.pdf', 'hello.txt', 'hi.txt', 'image.bmp', 'IMG_9684.JPG', 'IMG_9687.JPG', 'ppt.pptx',
         'Untitled.png']
```


jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (autosaved)
Log
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python
Run
Code

```
found a pdf file please wait while the system sorts these files.
moved pdf image
found a txt file please wait while the system sorts these files.
moved txt file
found a txt file please wait while the system sorts these files.
moved txt file
found an Unrecognised file please wait while the system sorts these files.
moved unknown files to seperate folder
found a jpg file please wait while the system sorts these files.
moved JPG file
found a jpg file please wait while the system sorts these files.
moved JPG file
found an Unrecognised file please wait while the system sorts these files.
moved unknown files to seperate folder
found a png file please wait while the system sorts these files.
moved png image
We are happy to inform you that we have transfered all the files into the correct folder.

In [16]: files!=None

Out[16]: True

In [17]: print(files)

['derivatives.pdf', 'hello.txt', 'hi.txt', 'image.bmp', 'IMG_9684.JPG', 'IMG_9687.JPG', 'ppt.pptx', 'Untitled.png']
```

WORST CASE:


jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

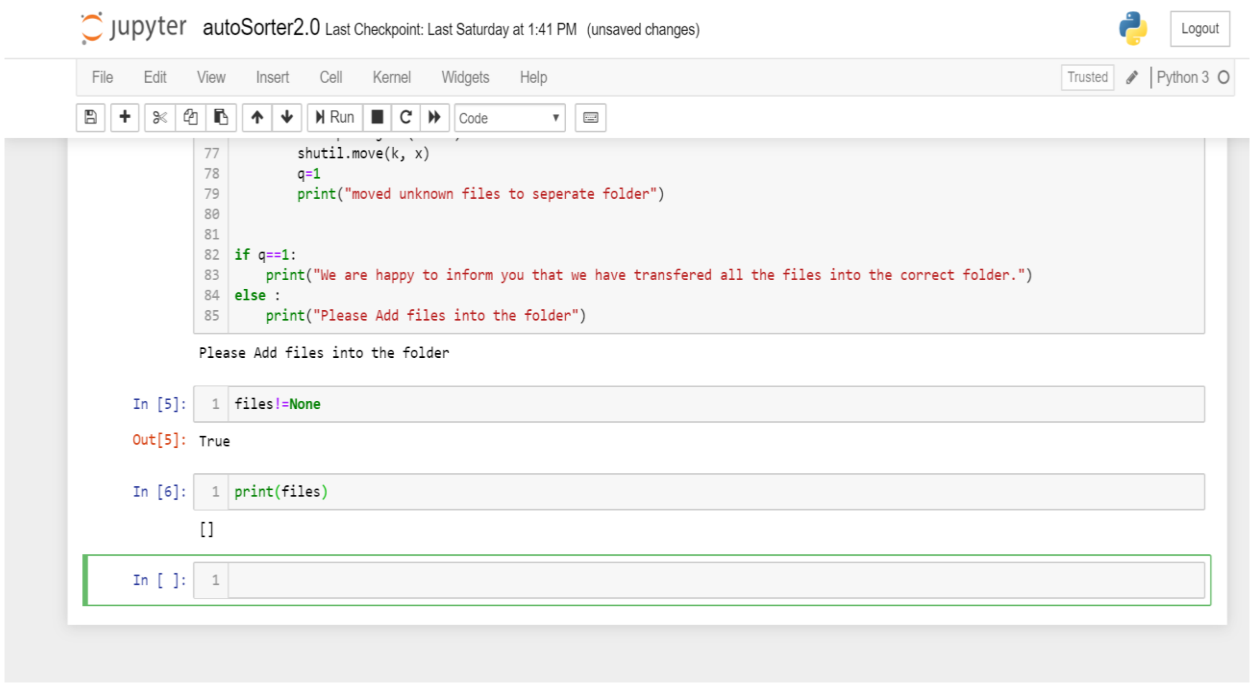
In [1]:
```python
import os
import re
import time as t
import shutil
```

In [2]:
```python
src=r"C:/Users/Manni/Desktop/autosorter/data/"
dst=r"C:/Users/Manni/Desktop/autosorter/files/"
files=os.listdir(src)
```

In [3]:
```python
print("Processing.....Please wait")
t.sleep(5)
print("Files that were detected : " , files)
```

```
Processing.....Please wait
Files that were detected :  []
```

In [4]:
```python
q=0
for i in files:
    if re.search("\w*?.txt",i):
        print("found a txt file please wait while the system sorts these files.")
        t.sleep(3)
        x=os.path.join(dst+"txt")
        try :
            os.mkdir(x)
        except :
```


jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```python
        shutil.move(k, x)
        q=1
        print("moved unknown files to seperate folder")


if q==1:
    print("We are happy to inform you that we have transfered all the files into the correct folder.")
else :
    print("Please Add files into the folder")
```

```
Please Add files into the folder
```

In [5]:
```python
files!=None
```

Out[5]: True

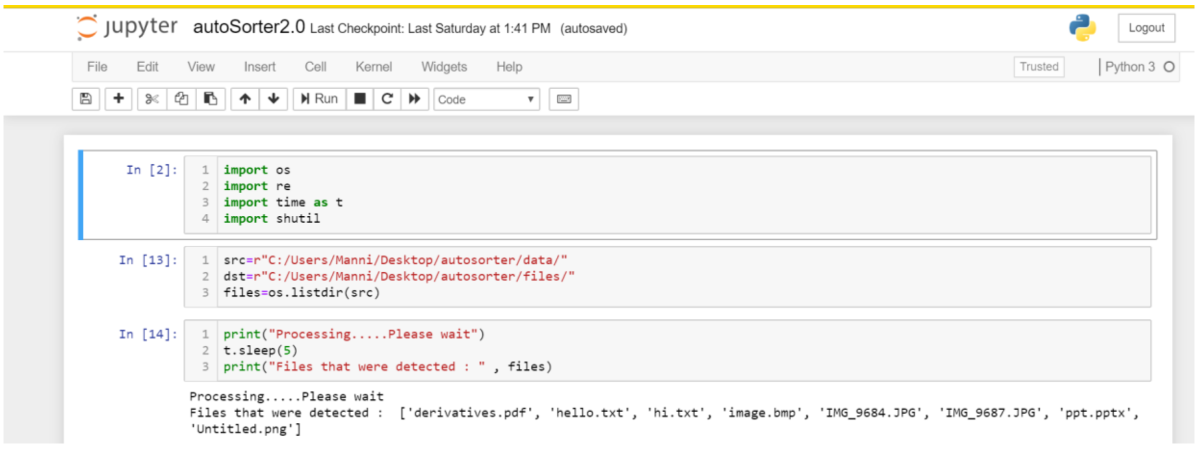
In [6]:
```python
print(files)
```

```
[]
```

In [ ]:

APPENDIX B

SOURCE CODE

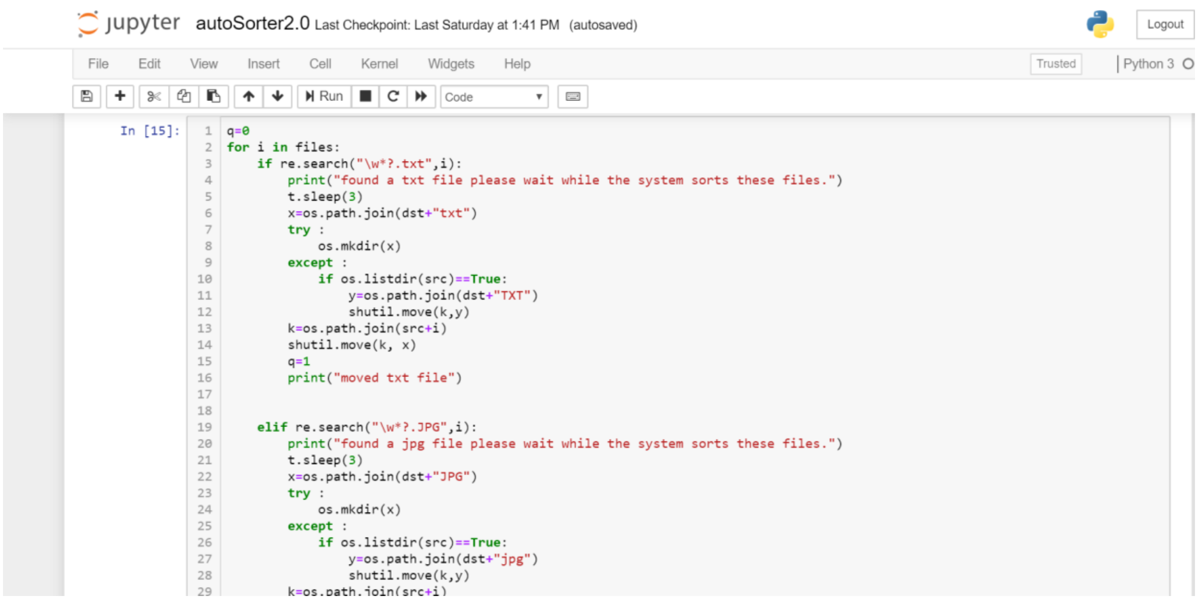

jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [2]:
import os
import re
import time as t
import shutil
```

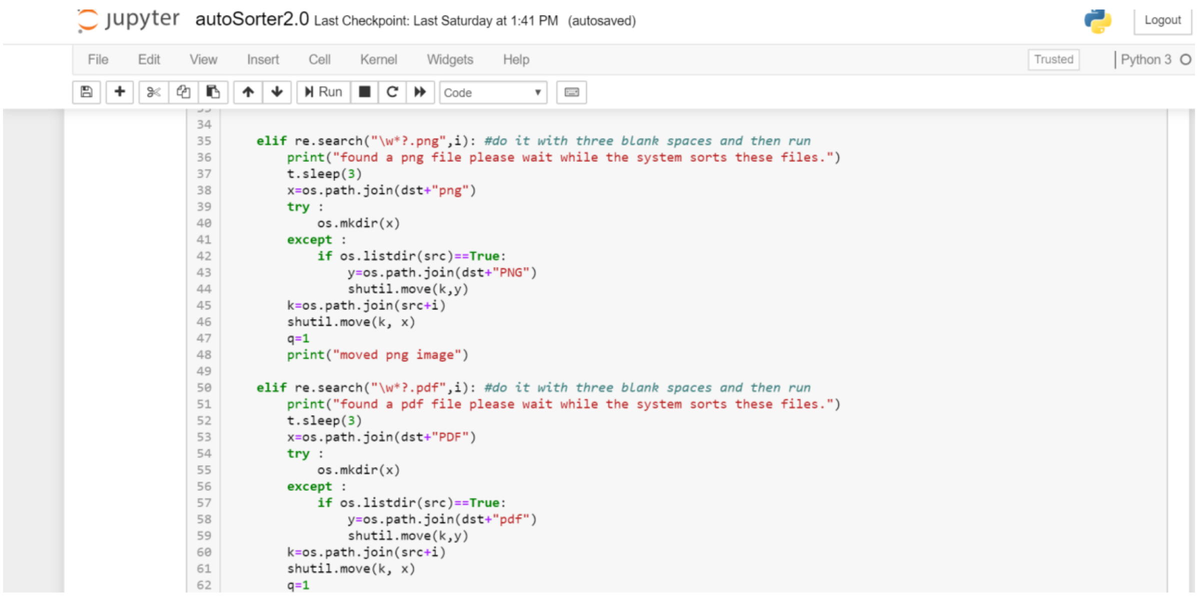
```
In [13]:
src=r"C:/Users/Manni/Desktop/autosorter/data/"
dst=r"C:/Users/Manni/Desktop/autosorter/files/"
files=os.listdir(src)
```

```
In [14]:
print("Processing.....Please wait")
t.sleep(5)
print("Files that were detected : " , files)
```

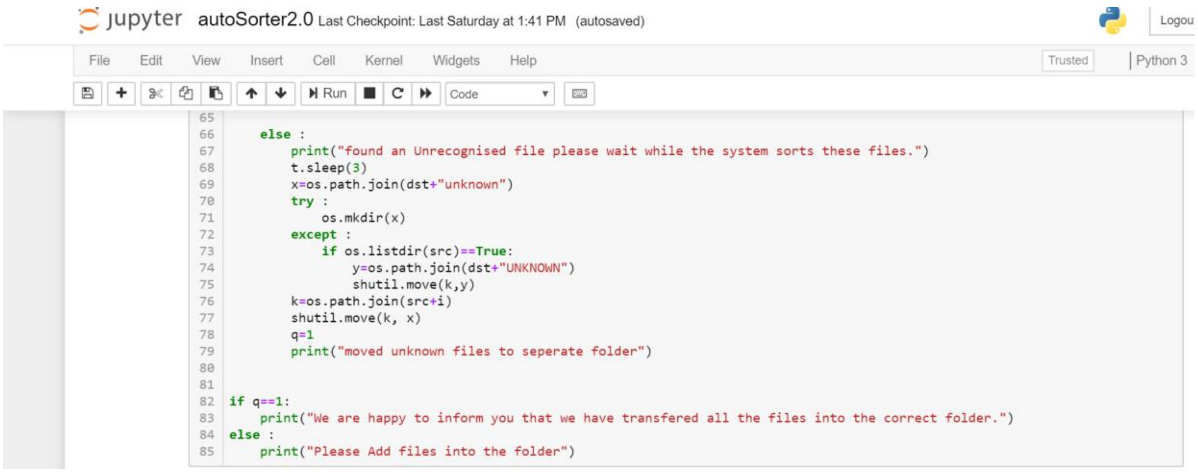
```
Processing.....Please wait
Files that were detected :  ['derivatives.pdf', 'hello.txt', 'hi.txt', 'image.bmp', 'IMG_9684.JPG', 'IMG_9687.JPG', 'ppt.pptx',
'Untitled.png']
```


jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [15]:
q=0
for i in files:
    if re.search("\w*?.txt",i):
        print("found a txt file please wait while the system sorts these files.")
        t.sleep(3)
        x=os.path.join(dst+"txt")
        try :
            os.mkdir(x)
        except :
            if os.listdir(src)==True:
                y=os.path.join(dst+"TXT")
                shutil.move(k,y)
        k=os.path.join(src+i)
        shutil.move(k, x)
        q=1
        print("moved txt file")


    elif re.search("\w*?.JPG",i):
        print("found a jpg file please wait while the system sorts these files.")
        t.sleep(3)
        x=os.path.join(dst+"JPG")
        try :
            os.mkdir(x)
        except :
            if os.listdir(src)==True:
                y=os.path.join(dst+"jpg")
                shutil.move(k,y)
        k=os.path.join(src+i)
```


jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```python

    elif re.search("\w*?.png",i): #do it with three blank spaces and then run
        print("found a png file please wait while the system sorts these files.")
        t.sleep(3)
        x=os.path.join(dst+"png")
        try :
            os.mkdir(x)
        except :
            if os.listdir(src)==True:
                y=os.path.join(dst+"PNG")
                shutil.move(k,y)
        k=os.path.join(src+i)
        shutil.move(k, x)
        q=1
        print("moved png image")

    elif re.search("\w*?.pdf",i): #do it with three blank spaces and then run
        print("found a pdf file please wait while the system sorts these files.")
        t.sleep(3)
        x=os.path.join(dst+"PDF")
        try :
            os.mkdir(x)
        except :
            if os.listdir(src)==True:
                y=os.path.join(dst+"pdf")
                shutil.move(k,y)
        k=os.path.join(src+i)
        shutil.move(k, x)
        q=1
```


jupyter autoSorter2.0 Last Checkpoint: Last Saturday at 1:41 PM (autosaved)
Logou
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```python

    else :
        print("found an Unrecognised file please wait while the system sorts these files.")
        t.sleep(3)
        x=os.path.join(dst+"unknown")
        try :
            os.mkdir(x)
        except :
            if os.listdir(src)==True:
                y=os.path.join(dst+"UNKNOWN")
                shutil.move(k,y)
        k=os.path.join(src+i)
        shutil.move(k, x)
        q=1
        print("moved unknown files to seperate folder")


if q==1:
    print("We are happy to inform you that we have transfered all the files into the correct folder.")
else :
    print("Please Add files into the folder")
```