Technological Educational Institute of Crete

Department of Informatics Engineering

Software Engineering

BSC. THESIS

SUPERVISION OF DATA TRANSFER WITH PYTHON

Student: Fifli Konstantina

AM: 2422

Supervisor Professor: Papadakis Nikolaos

Heraklion, 2016

Acknowledgment

I wish to express my sincere thanks to Mr. Papadakis, my thesis supervisor, for providing me with all the necessary information and guiding me during the whole process. I take this opportunity to express gratitude to all the Department faculty members for their technical support. Principally, special thanks to my parents, Kyriakos and Kondulia Fifli for the unceasing encouragement and the continuous economic support and inspiration. Finally, I would like to express my gratitude to all friends who have accompanied me during all those academic years.

Σύνοψη

Σήμερα, υπάρχουν πολύ περισσότεροι άνθρωποι που αλληλεπιδρούν με πληροφορίες. Το ποσό των ψηφιακών πληροφοριών αυξάνεται κατά δέκα φορές κάθε πέντε χρόνια. Κάθε μέρα, δημιουργούμε 2,5 τετράκις εκατομμύρια bytes δεδομένων - το 90% των δεδομένων στον κόσμο σήμερα έχει δημιουργηθεί τα τελευταία δύο χρόνια μόνο. Τα παραπάνω στοιχεία προέρχονται από παντού: αισθητήρες που χρησιμοποιούνται για τη συλλογή πληροφοριών κλίματος, δημοσιεύσεις σε δικτυακούς τόπους κοινωνικών μέσων ενημέρωσης, ψηφιακές φωτογραφίες και βίντεο, καθώς και τα σήματα GPS κινητών τηλέφωνων, για να αναφέρουμε μερικά. Ο κόσμος περιέχει αφάνταστα τεράστια ποσότητα ψηφιακών πληροφοριών που γίνεται αχανές όλο και πιο γρήγορα. Επιπλέον, να εξασφαλίσουμε την ασφάλεια των δεδομένων και την προστασία τους γίνεται όλο και πιο δύσκολο.

Η δημιουργία αντιγράφων ασφαλείας των δεδομένων του υπολογιστή είναι κρίσιμη. Είναι κάτι που πρέπει να ληφθεί σοβαρά υπόψη, για παράδειγμα, εταιρείες σήμερα θα μπορούσαν να οδηγηθούν σε πτώχευση από εάν χάσουν τα δεδομένα τους. Η δημιουργία αντίγραφων ασφαλείας (Backup) είναι μια διαδικασία που είναι πλέον υποχρεωτική και πρέπει να εκτελεστεί σωστά και αποτελεσματικά. Όμως, εξαιτίας πολλών προβλημάτων, τα αντίγραφα ασφαλείας μπορεί να μην πραγματοποιηθούν με επιτυχία.Η παρουσία ενός προγράμματος, το οποίο επιτρέπει την ανίχνευση διπλοαντιγράφων δεδομένων, καθώς επίσης, παρέχει την ασφάλεια ενός υψηλού επίπεδου μεταφοράς δεδομένων, αποτελεί ένα χρήσιμο εργαλείο για τη διαχείριση των δεδομένων.

Ένα πρόγραμμα που είναι σε θέση να μειώσει δραματικά την άσκοπη χρήση επιπλέον χώρου αποθήκευσης και επιπλέον, βοηθά τους ανθρώπους να ελαχιστοποιήσουν το χρόνο που δαπανούν για να οργανώσουν τα δεδομένα τους και κατά γενική ομολογία, απαλλάσσει τους πελάτες από ένα μεγάλο φόρτο εργασίας

Abstract

Nowadays, there are much more people who interact with information. The amount of digital information increases tenfold every five years. Every day we create 2.5 quintillion bytes of data, so much that 90% of the data in the world today has been created in the last two years only. Data come from everywhere: sensors used to gather climate information, posts on social media websites, digital pictures, videos, purchase transaction records, and cell phone GPS signals to name a few. The world contains an unimaginably vast amount of digital information that is currently getting bigger and bigger every day.

Moreover, ensuring data security and protecting them is becoming harder as information are shared even more widely around the world. Backing up computer's data is critical and is something that should not be taken lightly, for instance companies today could go bankrupt by losing all their data. Backup is a process that is now mandatory and that needs to be performed properly and efficiently.

The presence of a program allowing the detection of duplicated data as well as providing the insurance of a high performed transfer, constitutes a useful tool for insuring that data are correctly transferred between two locations, and reduce dramatically the unnecessarily occupying extra space of storage. In addition, it helps people to minimize the time spend on organizing their data and admittedly exempts people of a significant workload.

Content

Chapter 1. Introduction

1.1 Summary

In the present BSc. thesis is designed, developed and implemented, an autonomous software program for supervising and securing any data transfer. The purpose of this program is to better organize data and transfer them in a safer and harmless way. It can be benefic to a large range of users like companies, big sized or small sized, up to a single person who owns a computer. Briefly, this program is designed to provide the opportunity to check for duplicated data of any type as well as to detect data that have been modified and/or altered while transferring from a storage to another.

This program is coded with Python scripting language and developed through the PyCharm editor. Also are used, the Markup Language HTML helped by CSS for an appropriate display of the results.

1.2 Motivation for conducting project

The evolution of technology has influenced our lives and our "technological" habits. Consequently, the amount of data which we produce daily is increased. As the use of the internet is broad and the technological achievements greatly multiply, several amounts of data are produced and stored in local disks, removable disks, cloud, and servers. Sometimes we even have duplications of the same data which is unnecessarily occupying extra space of our storage. At one point, this data redundancy creates the unavoidable need to delete some data or to alternatively transfer it onto another storage. However, due to several reasons, the transfer of data might not happen successfully.

The presence of a program, which allows the detection of the unnecessarily duplicated data, provides the insurance of a high performed transfer, constitutes a useful tool for managing data. In addition, beyond its use case, it helps people to minimize the time spend on organizing their data and admittedly, exempts clients of a significant workload.

As part of the requirements which are referenced above, is developed a program which is the main subject of the current thesis.

1.3 Main subject

The main goal of this thesis is to compare data and supervise its transfer. We need to frequently delete unnecessary data which are duplicated or just updated and modified.

File comparison is an important, and most likely integral, part of file synchronization and backup. In backup methodologies the issue of data corruption is crucial. Corruption occurs without warning and without our knowledge, therefore, once it is noticed then it's too late to recover the missing parts. Usually, the only way to know for sure that a file has been corrupted is when it is opened by the user. Barring that, the user must use a comparison tool to at least recognize that a difference has been made. Therefore, all file synchronization or backup programs must include file comparison to be useful and trustworthy.

Below are summarized the functions performed by the program.

- Compares any type of format (ex. Microsoft Word, Excel, PowerPoint, PDF, Text, DOCX)
- Track file deletions (files deleted on either side)
- Track file conflicts (files changed on both sides)
- Detect and compare hidden files/folders
- Detect Size difference
- Detect Digest difference
- Follow and compare symbolic links
- Compare histogram data
- Check both ASCII & Binary
- Works with all media: hard disks, USB disks, NAS, flash drives, DVD-RW, CD-RW, etc.
- Works between computers, over Network, LAN, VPN

- 5 Reconciliation Methods: Synchronization, Backup, Mirroring, Replication, Consolidation
- No limit on data size or file size

Furthermore, it will list all the export comparison results, in a generated TXT file and in an HTML report.

Below are summarized what will be performed by the HTML report and the TXT file.

- The paths of the missing folders/files
- The paths of the modified folders/files
- The paths of duplicated files
- The numbers of items compared
- The real size, in bytes and in GB

1.4 Structure

Below is described the current project's structure.

Chapter 2 is describing the expected research on the subject in order to define the methodology that will be follow. In chapter 3 is covering the implementation phase, which is divided into two subcategories, the implementation phase itself and the test phase. The consequent results of the project can be found in chapter 4 enrichened with figures in order to give plain examples. Shortly before the end, chapter 5 is composed of the conclusions and the possible future extensions. Chapter 6 contains the bibliography and in chapter 7 are listed the appendixes.

Chapter 2. Research & Methodology

2.1 Programming Language

To begin with, the idea of creating a tool which can compare data was not an innovation. By doing a quick research, we can easily find that there are plenty of software offering the possibility to compare data. Despite all of these, the decision to build a totally new tool was taken. A tool which will not cost anything and which will meet all the user's expectations.

Starting with the research of the programming language which will be used to develop the idea, is usually the first thing to do. After taking a deep and careful general look at many programming languages, Python was the one catching the interest. To justify this choice the following section covers an introduction about Python, firstly, and secondly a comparison between Python and few others programming languages.

2.1.1 What is Python? Executive Summary

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, makes it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn as the syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

Often, programmers choose work with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy, a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source, the fast edit-test-debug cycle makes this simple approach very effective.

2.1.2 Comparing Python to Other Languages

Python is often compared to other interpreted languages such as Java, JavaScript, and Perl. Comparisons to C can also be enlightening. In this section, will briefly compare Python to each of these languages. These comparisons are focused on language issues only. In practice, the choice of a programming language is often dictated by other real-world constraints such as cost, availability, training, and prior investment, or even emotional attachment. Since these aspects are highly variable, it seems a waste of time to consider them much for this comparison.

- Java

Python programs are generally expected to run slower than Java programs, but they also take much less time to develop. Python programs are typically 3-5 times shorter than equivalent Java programs. This difference can be attributed to Python's built-in high-level data types and its dynamic typing. For example, a Python programmer wastes no time declaring the types of arguments or variables. Also Python's powerful polymorphic list and dictionary types, for which rich syntactic support is built straight into the language, find a use in almost every Python program. Because of the run-time typing Python's run time must work harder than Java's. For example when evaluating the expression a+b, it must first inspect the object a and b to find out their type which is not known at compile time. It then invokes the appropriate addition operation which may be an overloaded user-defined method. Java, on the other hand, can perform an efficient integer or floating point addition but requires variable declarations for a and b and does not allow overloading of the + operator for user-defined classes instances.

For these reasons, Python is much suited as a "glue" language while Java is better characterized as a low-level implementation language. In fact, the two together make an excellent combination. Components can be developed in Java and combined to form applications in Python. Python can also be used to prototype components until their design can be "hardened" in a Java implementation. To support this type of development, a Python implementation written in Java is under development which allows calling Python code from Java and vice versa. In this implementation, Python source code is translated to Java bytecode (with help from a run-time library to support Python's dynamic semantics).

❖ JavaScript

Python's "object-based" subset is roughly equivalent to JavaScript. Like JavaScript (and unlike Java), Python supports a programming style that uses simple functions and variables without engaging in class definitions. However, for JavaScript, that's all there is. Python, on the other hand, supports writing much larger programs and better code reuse through a true object-oriented programming style where classes and inheritance play an important role.

❖ Perl

Python and Perl come from a similar background (UNIX scripting which both have long outgrown) and sport many similar features but have a different philosophy. Perl emphasizes support for common application-oriented tasks, e.g. by having built-in regular expressions, file scanning and report generating features. Python emphasizes support for common programming methodologies such as data structure design and object-oriented programming, and encourages programmers to write readable (and thus maintainable) code by providing an elegant but not overly cryptic notation. As a consequence, Python comes close to Perl but rarely beats it in its original application domain; however Python has an applicability well beyond Perl's niche.

❖ C++

Almost everything said for Java also applies for C++, just more so: where Python code is typically 3-5 times shorter than equivalent Java code, it is often 5-10 times shorter than equivalent C++ code. Anecdotal evidence suggests that one Python programmer can finish in two months what two C++ programmers can't complete in a year. Python shines as a glue language, used to combine components written in C++.

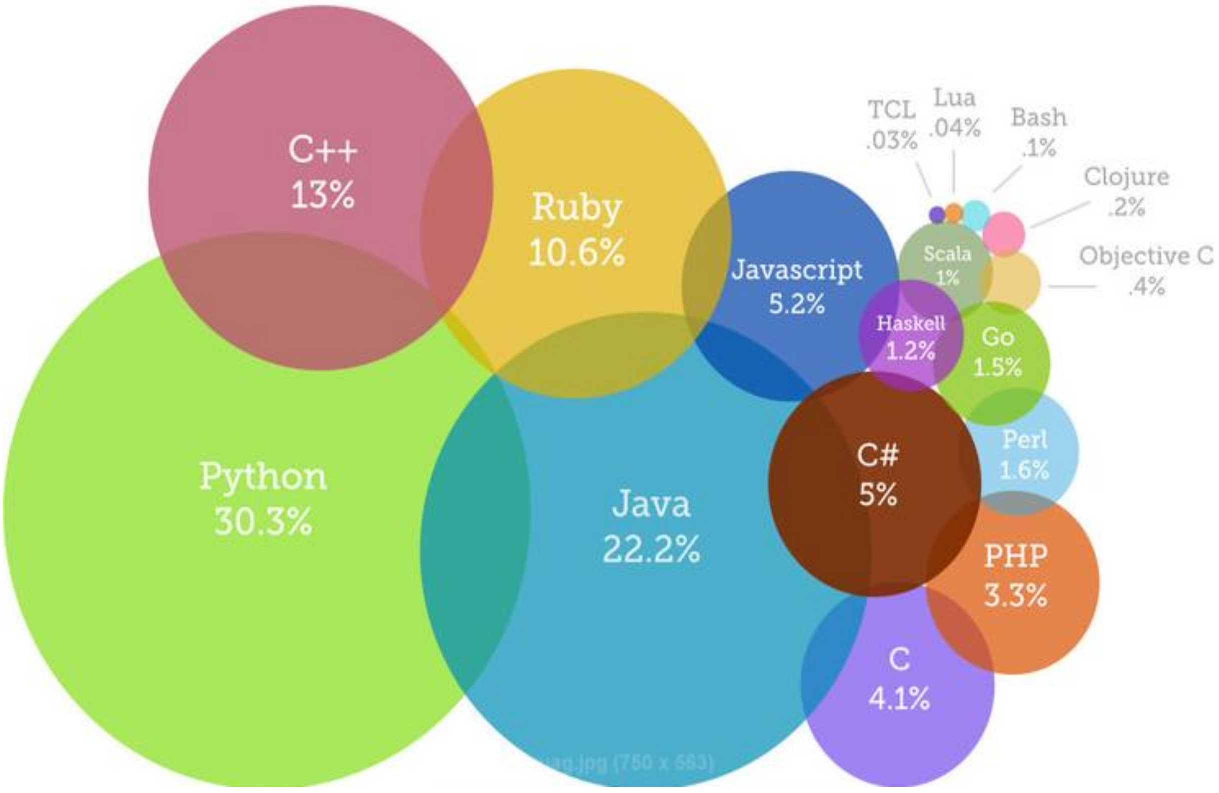


Figure 1. Most popular programming Language

2.1.3 Python as Programming Language

Due to the above reasons, Python was chosen as Programming Language. To summarize it all:

- ❖ Python is robust

It's solid and powerful. Python has a relative small quantity of lines of code, which makes it less prone to issues, easier to debug, and more maintainable. It's also very fast.

- ❖ Python is flexible

Because it wasn't originally created to answer a specific need, Python isn't driven by templates or specific APIs and is, therefore, well-suited to the rapid development of all kinds of applications.

- ❖ Python is easy to learn and use

"Python, in particular, emerges as a nearly ideal candidate for a first programming language", says *John M. Zelle, in the Department of Mathematics, Computer Science, and Physics at Wartburg College in Iowa.*

- ❖ Python is free

Since Python is an open source programming language, it immediately reduces up-front project costs by leveraging Python in the development projects.

In conclusion, Python can be used in a variety of situations, both online and offline. Here are just a few interesting places where Python is used:

- ❖ Google uses python in its spiders.
- ❖ NASA uses Python in its Integrated Planning System as the standard scripting language at Johnson Space Center.
- ❖ Red Hat uses Python for Red Hat Linux's installer (anaconda) and configuration utilities.
- ❖ IBM uses Python to create the business practice logic for factory tool control applications at IBM East Fishkill.

- The CIA built its website in Python with Zope.
- Walt Disney Feature Animation uses Python to add script ability to their animation production system.

2.1.4 Python Versions

As Python is chosen as Programming Language, another question came up. Should be used Python 2 or Python 3 for the development activity? Unlike other programming Languages, Python has introduced 2 different versions of itself, which are not totally compatible to each other. The state of things essential is as follow:

- Python 2.7 has been the standard for a long time.
- Python 3 introduced major changes to the language, which many developers are unhappy with.
- Python 2.7 will receive necessary security updates until 2020.
- Python 3 is continually evolving like Python 2 did in years past.

Figure 2. Versions of Python

After taking a closer look at the two different versions of the language, Python 3 is a nicer and more consistent language but there is very limited third-party module support for it. Python 2 version has been around for longer time, which can be an advantage, but not all the libraries available for Python 2 have been ported to Python 3.

As Python 2.7 version is used for a longer time and in many projects, it is going to be the version used during this project.

2.2 The Development Environment

In continue, after choosing the Programming Language and the specific version of it, the next step is to choose the Development Environment. The Language which is used and its interpreters or compilers are the only tools necessary to develop software. Another important matter is the Programming Environment. Unlike some Languages, where the choices are limited, there are one or two obviously superior options, Python has no "standard" tool as Python developers can use any editors and IDEs from a wide range of choice. Choosing the right tool for that project is not difficult but it is not to be taken lightly. Selection of the right editor can greatly influence productivity and efficiency of Python programming.

After making research around, there are plenty of options. Some people still prefer a basic text editor that can be extended with features like syntax highlighting and autocomplete. But a lot of power users working on large projects with complex code bases prefer an Integrated Development Environment (IDE) than the text editor and its terminal.

2.2.1 What is a Text Editor?

There are text editors that are made specifically for writing programming Languages. Commonly called programming text editors to highlight the difference, they are generally known simply as text editors. They still only deal with plain text files but they also have some handy features for programmers:

- ❖ Syntax Highlighting

Colors are used to highlight different parts of a program. It makes code easier to read and debug. For instance you could set up syntax highlighting so that Python keywords are blue, comments are green, string literals are orange and so on.

- ❖ Automatic Editing

Programmers format their programs so that blocks of code are indented together. This indentation can be done automatically by the editor.

❖ Compilation and Execution Commands

To save the programmer having to switch from the text editor to a terminal window these editors have the ability to compile and execute programs. Therefore, debugging can be done all in one place.

The chart below displays a survey of the most popular Text Editors for Python. For the survey were interviewed 100 Python specialists. *Survey "Which Code Editors Do Pythonists Use?" by Abder-Rahman Ali for SitePoint.* The most used editor is Sublime Text.

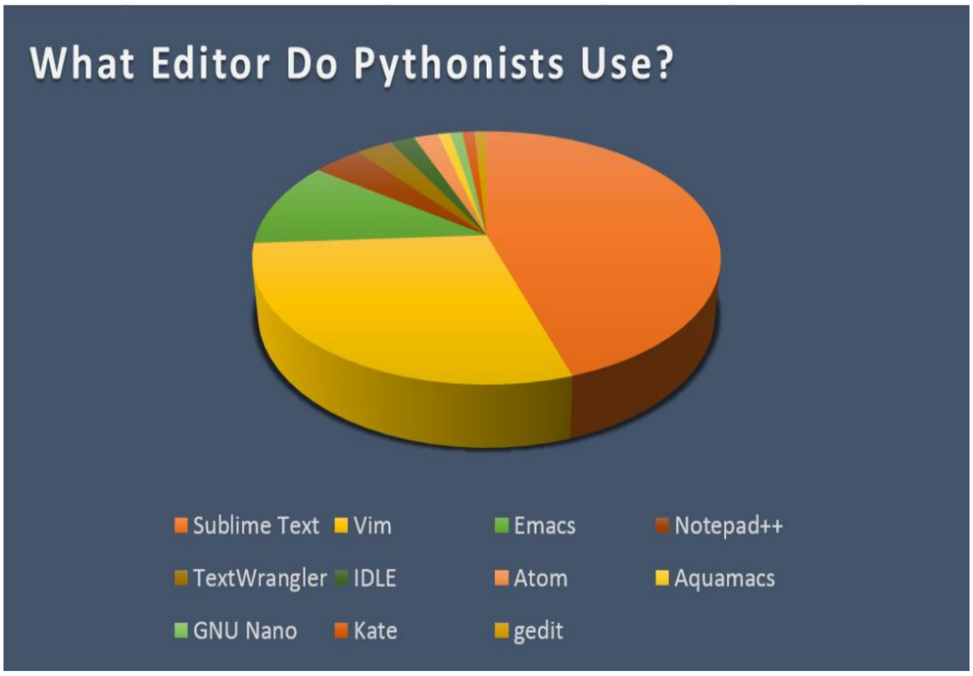


Figure 3. Text Editors

2.2.2 What is an IDE?

IDE stands for Integrated Development Environment. An integrated development environment (IDE) is a programming environment that has been packaged as an application program, typically composed of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder. They are powerful tools for programmers that offer all the features of a programming text editor and much more. The idea behind an IDE is to encompass everything a programmer could want to do in one application. Theoretically, it should allow them to develop programs faster.

There are so many features an IDE can contain but the following list contains only few selected ones. It should highlight how useful they can be to programmers:

- ❖ Automatic Code Completion

Whilst typing, the IDE can help by showing a list of possible options. For example, when using a String object a programmer might want to use one of its methods. As they type a list of methods, that they can pick one, will appear in a popup menu.

- ❖ Access Databases

To help connect applications to databases IDEs can access different databases and query data within them.

- ❖ GUI Builder

Graphical user interfaces can be created by dragging and dropping Swing components on to a canvas. The IDE automatically writes the code that creates the GUI.

- ❖ Optimization

As applications become more complex, speed and efficiency become more important. Profilers built into the IDE can highlight areas where the code could be improved.

- ❖ Version Control

Previous versions of source code files can be kept. It's a useful feature because a working version can be stored. If in the future it is modified then a new version can be created. If the modifications cause problems the file can be rolled back to the previous working version.

2.2.3 Text Editor Vs IDE

Eventually, IDE's and Text Editors are fundamentally different tools that each has their strengths and weaknesses. Concisely, they are presented further down.

Text Editor: Strengths

- Fast
- Easy to extend (macros, plugins)
- Text edit functions (Ex: sublime text 2 unending keyboard shortcuts)

Text Editor: Weaknesses

- Need to use another service to compile
- Low support for code completion (intelligence features)

IDE: Strengths

- Integrated testing
- Compilation
- Breakpoints/stepping through code
- Integration with other services (database views), automated class diagrams

IDE: Weaknesses

- Large memory footprint
- Cost

After taking into account the advantages and the disadvantages of both Text editors and IDEs and what each one is able to provide to a developer, using an Integrated Development Environment is preferable.

Consequently, searching for the most suitable IDE is required. While searching for the most popular IDEs for Python, plenty of options appear on the way. Eclipse, Eric, IDLE, Komodo, PyCharm, Spyder, Wing IDE are just a few of them. After reading the advantages and disadvantages of each and taking into consideration some opinions of experienced developers, resulted a list with the 3 most popular IDEs for Python.

- ❖ Pydev with Eclipse
- ❖ PyCharm
- ❖ Wing IDE

To conclude with, *“There are some very good options among IDEs: if you want a free one that works well, install Eclipse and PyDev; if you are willing to pay money, PyCharm and Wing IDE have similar capabilities and are both excellent IDEs.” written by Jason Fruit, author at Python Central.* PyCharm became the official Integrated Development Environment of this project.

2.2.4 PyCharm

- ❖ Key Facts

PyCharm is a Python IDE with a complete set of tools for productive development with Python programming language. In addition, the IDE provides high-class capabilities for professional Web development with Django framework.

- ❖ Key Benefits

Like others IDEs, PyCharm offers a smart code editor which understands the specificities of Python and offers remarkable productivity booster: automatic code formatting, code completion, refactoring, auto import, one-click code navigation, and more. Backed by advanced code analysis routines, these features make PyCharm a powerful tool in the hands of both professional Python developers and those who are just starting out with the technology.

- ❖ Key Features or https://www.jetbrains.com/pycharm/features/index.html

• Coding Assistance: smart and configurable editor with code completion, snippets, and various intention actions.

• Code Analysis: on-the-fly code syntax, error highlighting, intelligent inspections and one-click quick-fix suggestions to improve the code.

• Project Code Navigation: quick navigation from one file to another, from method to its declaration or usages and through classes' hierarchy.

• Python Refactoring: project-wide code modifications are painless with rename, extract method/superclass, introduce field/variable/constant, move and pull up/push down refactoring.

• Web Development with Django: rapid Web development with Django framework backed up with excellent HTML, CSS, and JavaScript editing facilities.

• Version Control Integration: Check-in, check-out, view diffs, merge, all in the unified VCS user interface for Mercurial, Subversion, Git, Perforce and other SCMs.

• Integrated Unit testing: Run a test file, a single test class, a method, or all tests in a folder. Results are presented in a special graphical test runner with execution statistics.

• Graphical Debugger for Python or Django applications and unit tests with breakpoints, stepping and frames view, watches and evaluate expressions.

• Google App Engine Support for creating Google App Engine applications and delegating routine deployment tasks to the IDE.

• Customizable & Extensible Environment with bundled Textmate, NetBeans, Eclipse & Emacs keyboard schemes, and Vi/Vim emulation plugin.

Figure 4. PyCharm

2.3 Methodology

As important as the research is the creation of a methodology that it should be followed, strictly. At the moment an important step is to elaborate the way of working on the project. Should be taken under consideration the allowed time to accomplish the project which is nearly six months. In the following figure is displayed the Gantt diagram which was designed at the beginning of the project. The Gantt Diagram helped to prevent delayed deadline and to provide extra time to deal with unanticipated situations. It also helps to keep an order on the project and to have a concrete plan.

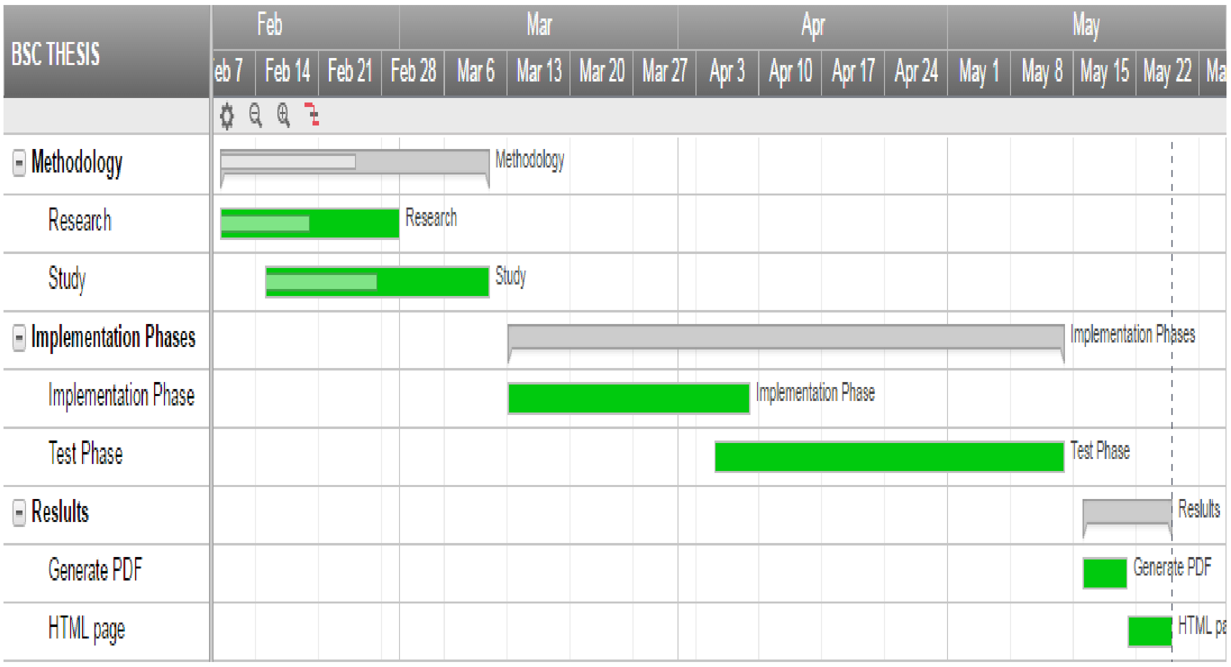


Figure 5. Gantt Diagram

The above figure is exhibiting the Gantt Diagram of the current thesis. The Gantt Diagram is displaying 3 categories: the methodology, the implementation phases and also the results. The methodology is composed of two subcategories, the research part and the personal time taken for training. In total, those two categories require a month to be accomplished. Then can be found the implementations phases which consist of the implementation phase itself and the phase called testing phase. As the implementations phases compose the real development of the project, they are given nearly two months, implementation time. The last category of the Gantt Diagram is called results and it includes the creation of a TXT file and of an HTML report for displaying the results of the implementation phase. More or less it will be needed 10 days according to the Gantt Diagram for the creation of both reports.

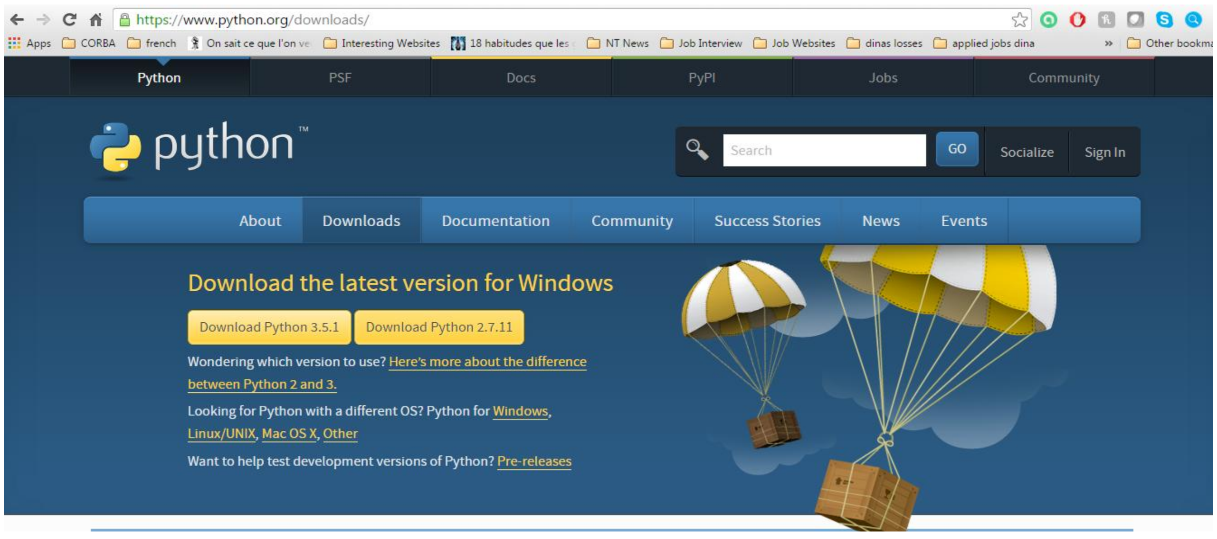
Chapter 3. Project implementation

3.1 Installations

In this chapter is developed step by step the implementation phase. The installations required are present, as well as, the development code. Should be mentioned that the operating system used for this project is Microsoft Windows 7.

3.1.1 Python installation

Firstly is present the installation of the programming language. In this case is Python 2.7 version. Python is found on the website: *https://www.python.org* and it can be downloaded and installed as below.



Download the latest version for Windows

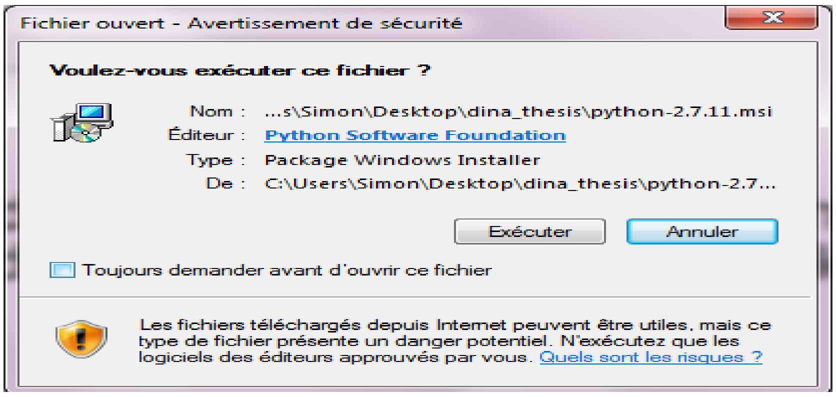
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

Figure 6. Download python

Select the desired version and follow the steps below:


Fichier ouvert - Avertissement de sécurité

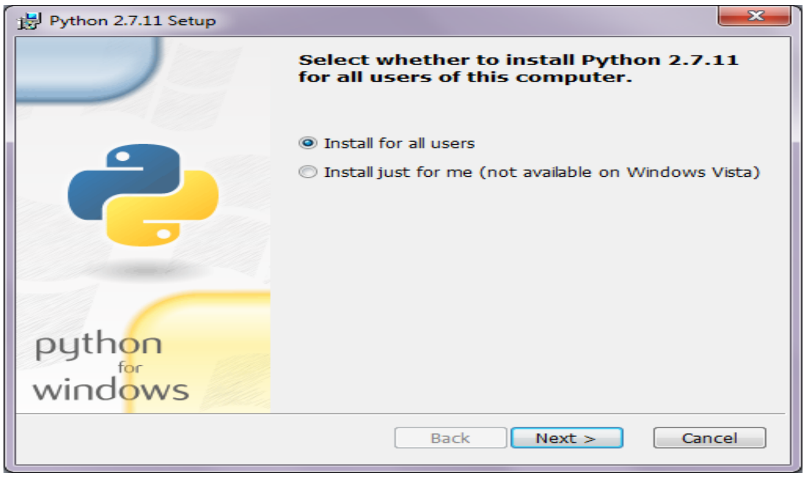
Voulez-vous exécuter ce fichier ?

Nom : ...s\Simon\Desktop\dina_thesis\python-2.7.11.msi
Éditeur : Python Software Foundation
Type : Package Windows Installer
De : C:\Users\Simon\Desktop\dina_thesis\python-2.7...

Toujours demander avant d'ouvrir ce fichier

Les fichiers téléchargés depuis Internet peuvent être utiles, mais ce type de fichier présente un danger potentiel. N'exécutez que les logiciels des éditeurs approuvés par vous. Quels sont les risques ?

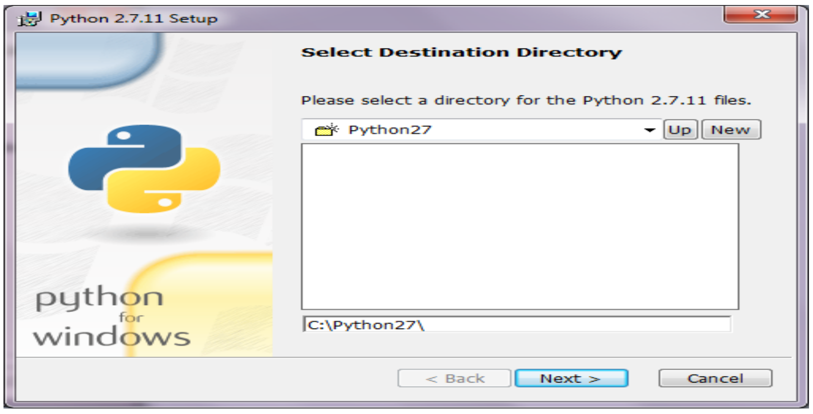
Figure 7. Step1: Python installation


Python 2.7.11 Setup

Select whether to install Python 2.7.11 for all users of this computer.

[✓ Install for all users]
Install just for me (not available on Windows Vista)

Figure 8. Step2: Python installation


Python 2.7.11 Setup

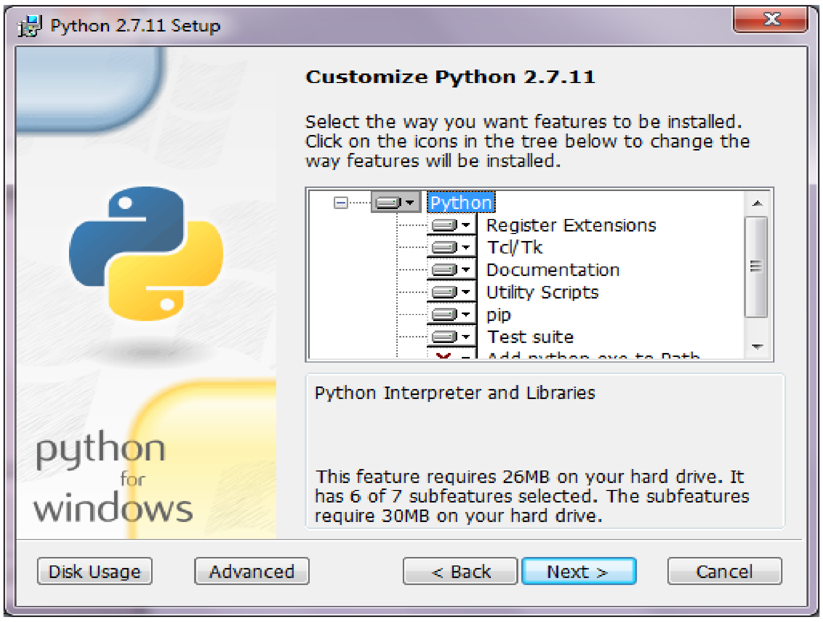
Select Destination Directory

Please select a directory for the Python 2.7.11 files.

Python27

C:\Python27\

Figure 9. Step3: Python installation



Customize Python 2.7.11

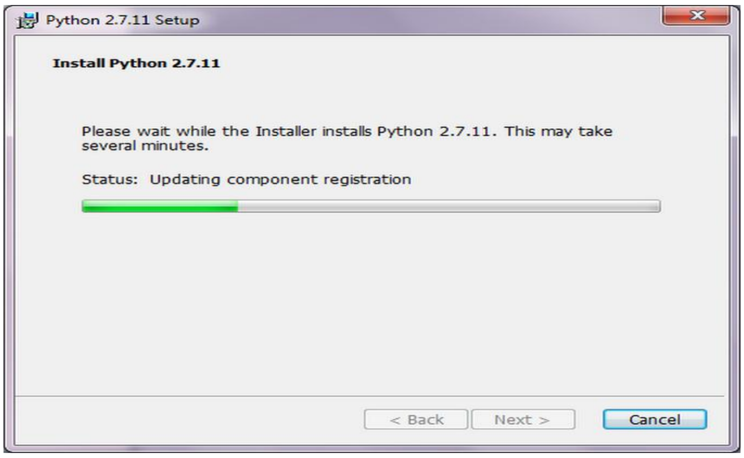
Select the way you want features to be installed.
Click on the icons in the tree below to change the way features will be installed.

Python
Register Extensions
Tcl/Tk
Documentation
Utility Scripts
pip
Test suite
Add python.exe to Path

Python Interpreter and Libraries

This feature requires 26MB on your hard drive. It has 6 of 7 subfeatures selected. The subfeatures require 30MB on your hard drive.

Figure 10. Step4: Python installation

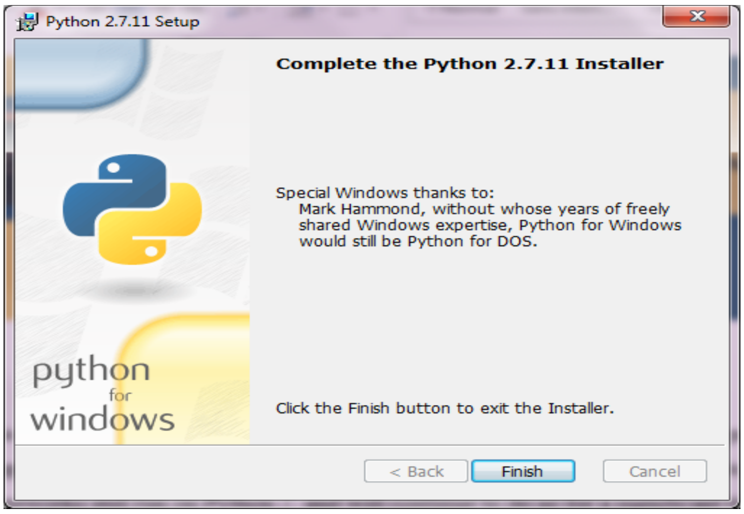


Install Python 2.7.11

Please wait while the Installer installs Python 2.7.11. This may take several minutes.

Status: Updating component registration

Figure 11. Step5: Python installation



Complete the Python 2.7.11 Installer

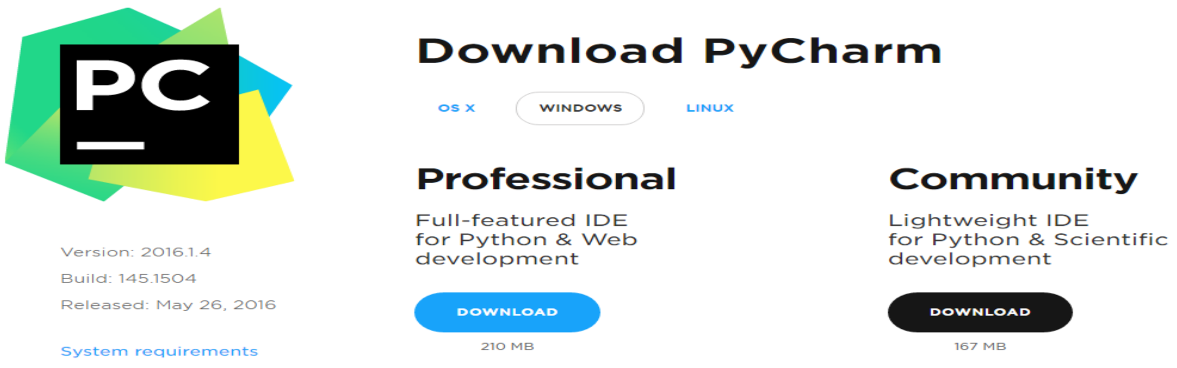
Special Windows thanks to:
Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

Click the Finish button to exit the Installer.

Figure 12. Step6: Python installation

3.1.2 IDE/PyCharm installation

Next step is to install the desired Integrated Development Environment. In this case the IDE is called PyCharm. Below are the steps to follow.



Download PyCharm

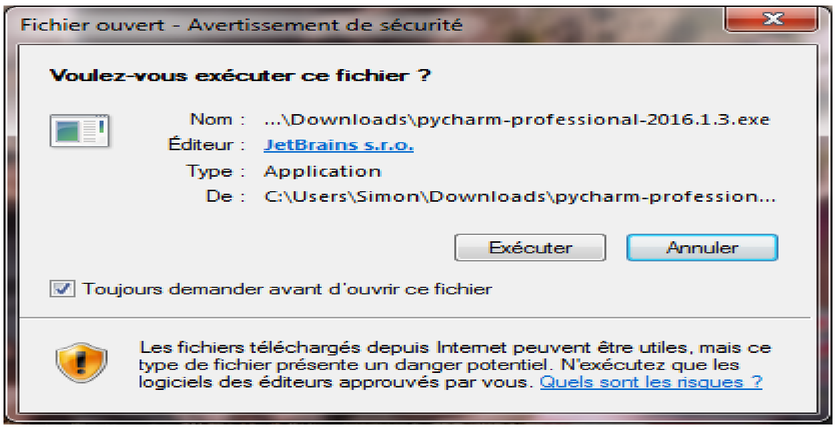
OS X | WINDOWS | LINUX

Professional
Full-featured IDE for Python & Web development
DOWNLOAD
210 MB

Community
Lightweight IDE for Python & Scientific development
DOWNLOAD
167 MB

Version: 2016.1.4
Build: 145.1504
Released: May 26, 2016
System requirements

Figure 13. PyCharm download



Fichier ouvert - Avertissement de sécurité

Voulez-vous exécuter ce fichier ?

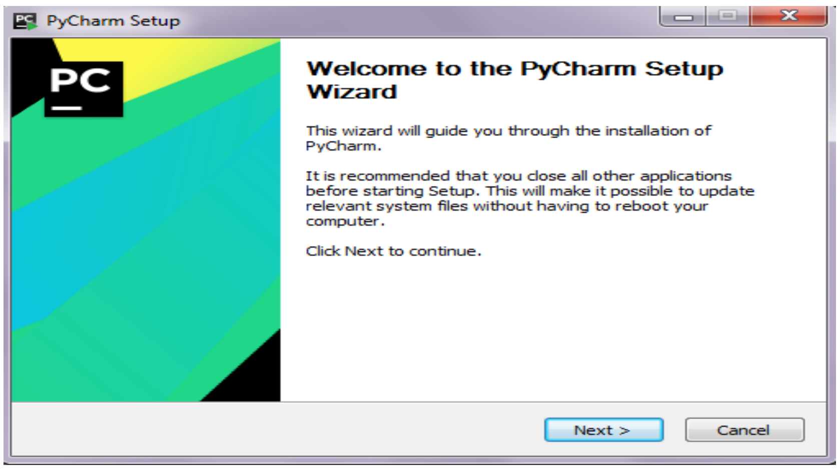
Nom : ...\Downloads\pycharm-professional-2016.1.3.exe
Éditeur : JetBrains s.r.o.
Type : Application
De : C:\Users\Simon\Downloads\pycharm-profession...

Exécuter | Annuler

☑ Toujours demander avant d'ouvrir ce fichier

Les fichiers téléchargés depuis Internet peuvent être utiles, mais ce type de fichier présente un danger potentiel. N'exécutez que les logiciels des éditeurs approuvés par vous. Quels sont les risques ?

Figure 14. Step1: pyCharm installation



PyCharm Setup

Welcome to the PyCharm Setup Wizard

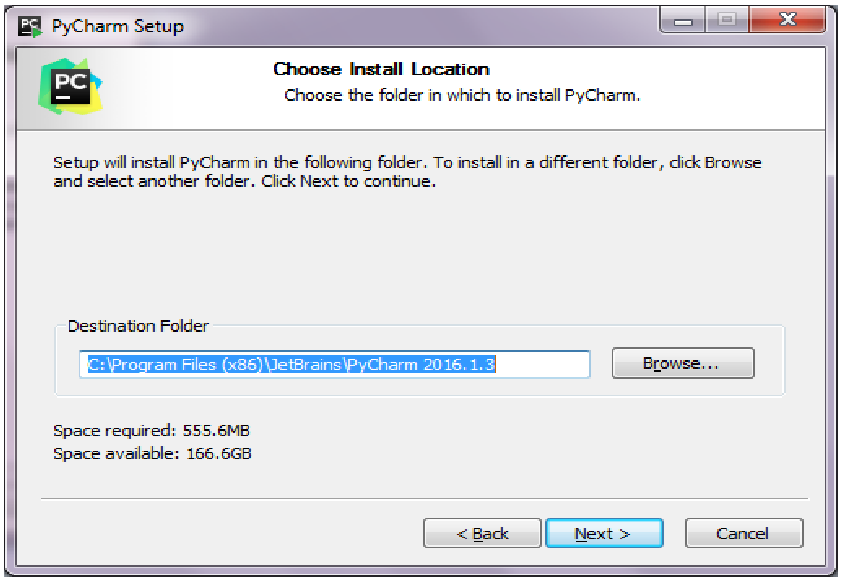
This wizard will guide you through the installation of PyCharm.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Next > | Cancel

Figure 15. Step2: pyCharm installation


PyCharm Setup

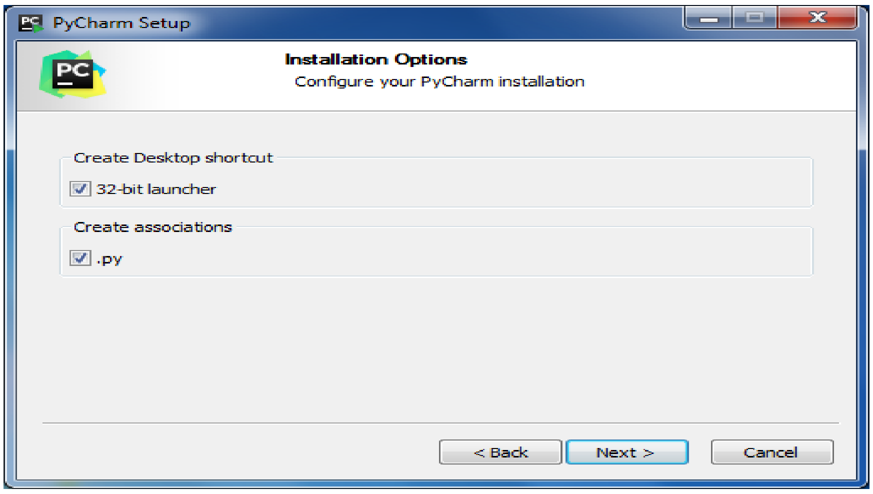
Choose Install Location
Choose the folder in which to install PyCharm.

Setup will install PyCharm in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder
C:\Program Files (x86)\JetBrains\PyCharm 2016.1.3

Space required: 555.6MB
Space available: 166.6GB

Figure 16. Step3: pyCharm installation

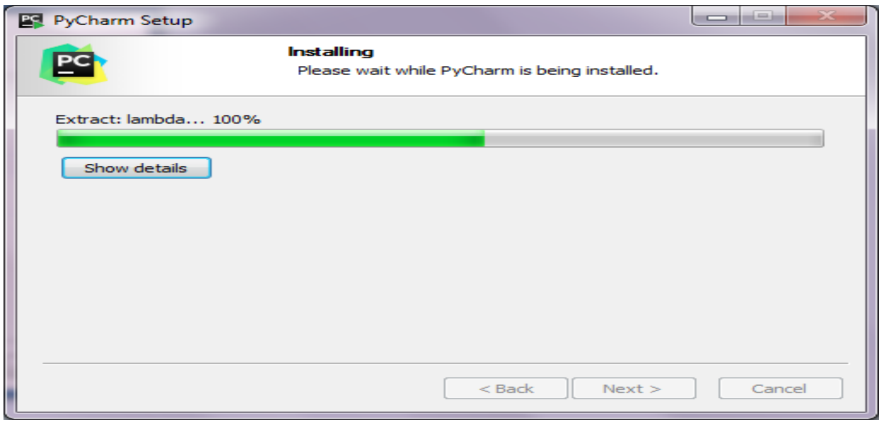

PyCharm Setup

Installation Options
Configure your PyCharm installation

Create Desktop shortcut
☑ 32-bit launcher

Create associations
☑ .py

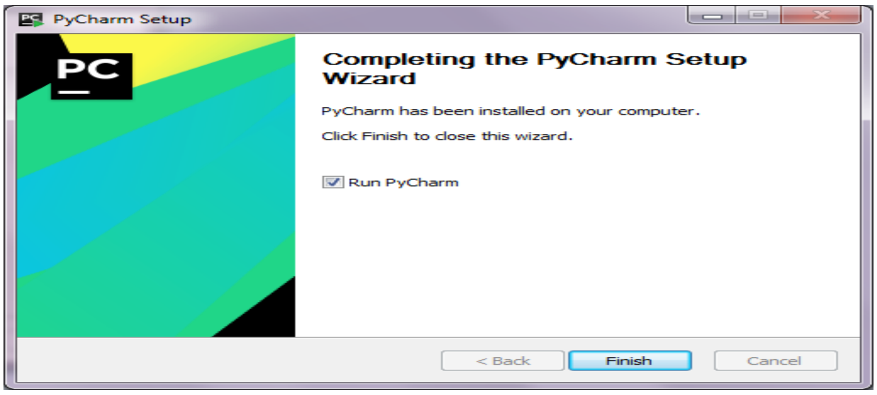
Figure 17. Step4: pyCharm installation


PyCharm Setup

Installing
Please wait while PyCharm is being installed.

Extract: lambda... 100%

Figure 18. Step5: pyCharm installation


PyCharm Setup

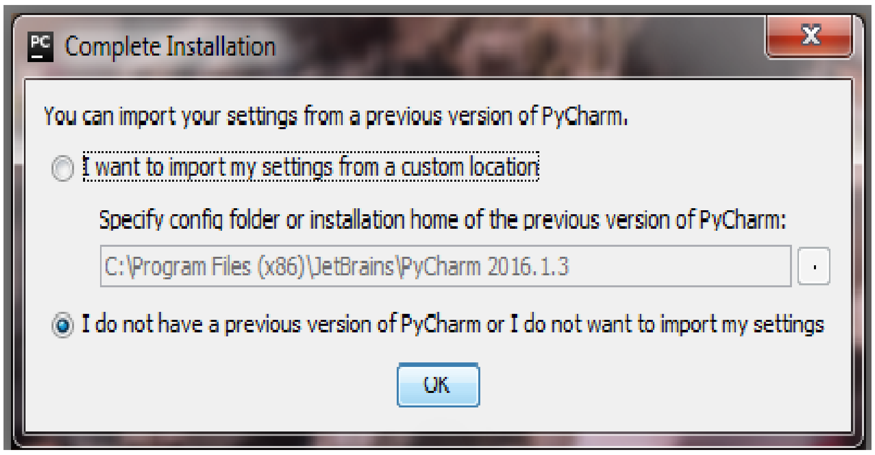
Completing the PyCharm Setup Wizard

PyCharm has been installed on your computer.

Click Finish to close this wizard.

[✓ Run PyCharm]

Figure 19. Step6: pyCharm installation


Complete Installation

You can import your settings from a previous version of PyCharm.

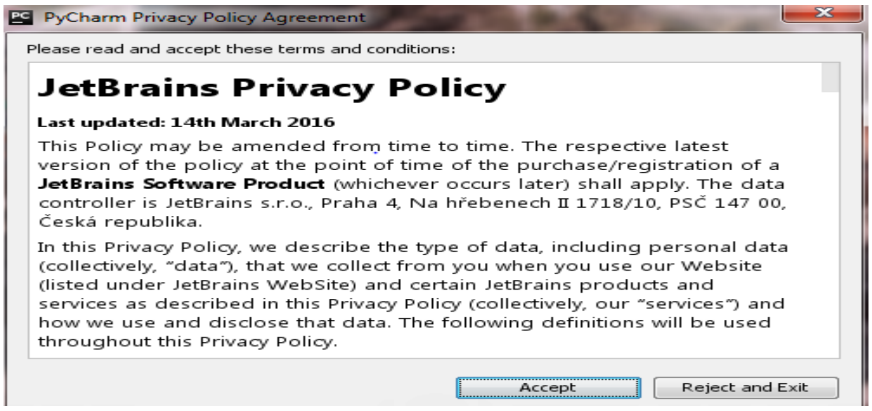
I want to import my settings from a custom location

Specify config folder or installation home of the previous version of PyCharm:

C:\Program Files (x86)\JetBrains\PyCharm 2016.1.3

[✓ I do not have a previous version of PyCharm or I do not want to import my settings]

Figure 20. Step7: pyCharm installation


PyCharm Privacy Policy Agreement

Please read and accept these terms and conditions:

JetBrains Privacy Policy

Last updated: 14th March 2016

This Policy may be amended from time to time. The respective latest version of the policy at the point of time of the purchase/registration of a **JetBrains Software Product** (whichever occurs later) shall apply. The data controller is JetBrains s.r.o., Praha 4, Na hřebenech II 1718/10, PSČ 147 00, Česká republika.

In this Privacy Policy, we describe the type of data, including personal data (collectively, "data"), that we collect from you when you use our Website (listed under JetBrains WebSite) and certain JetBrains products and services as described in this Privacy Policy (collectively, our "services") and how we use and disclose that data. The following definitions will be used throughout this Privacy Policy.

Figure 21. Step8: pyCharm installation

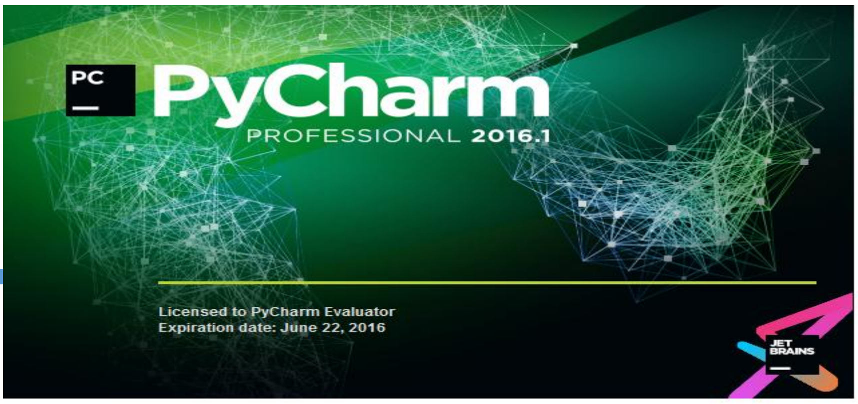
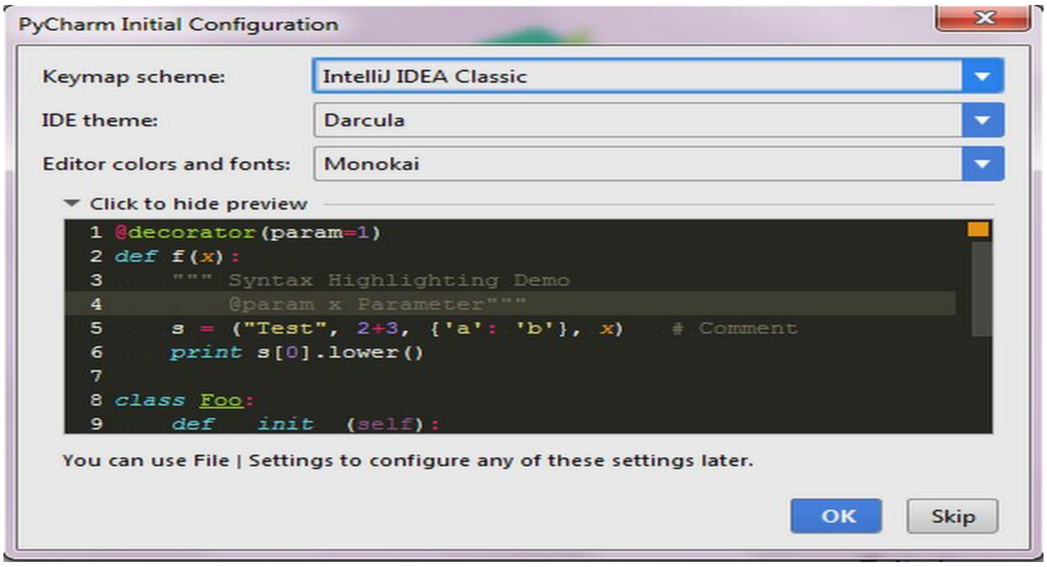


Figure 22. Step9: pyCharm installation

3.1.3 Configuring Python Interpreter

Within those simple steps, the installation of the pyCharm IDE is finished. As the installation is done, there are some configurations to take part afterwards. Those simple but necessary configurations are referenced below.


PyCharm Initial Configuration

Keymap scheme: IntelliJ IDEA Classic

IDE theme: Darcula

Editor colors and fonts: Monokai

Click to hide preview

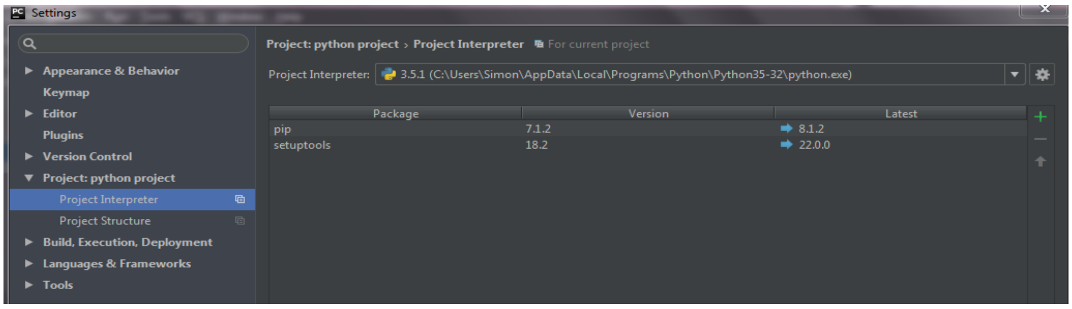
```
@decorator(param=1)
def f(x):
    """ Syntax Highlighting Demo
        @param x Parameter"""
    s = ("Test", 2+3, {'a': 'b'}, x)   # Comment
    print s[0].lower()

class Foo:
    def __init__(self):
```

You can use File | Settings to configure any of these settings later.

Figure 23. PyCharm theme

When the IDE is set up and working fine, the following steps are required to configure the interpreter before use:

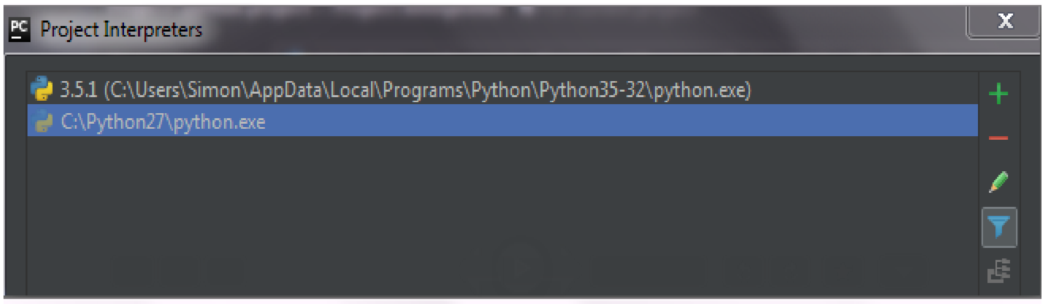
Open the Settings dialog box, and click Project Interpreter page. In the Projects pane, choose the desired project. For the selected project, choose SDK from the list of available Python interpreters and virtual environments. Scroll down and click on the option show all.



Project Interpreter: 3.5.1 (C:\Users\Simon\AppData\Local\Programs\Python\Python35-32\python.exe)

Package	Version	Latest
pip	7.1.2	➡ 8.1.2
setuptools	18.2	➡ 22.0.0

Figure 24. Step2 configure interpreter



3.5.1 (C:\Users\Simon\AppData\Local\Programs\Python\Python35-32\python.exe)
C:\Python27\python.exe

Figure 25. Step3 configure interpreter

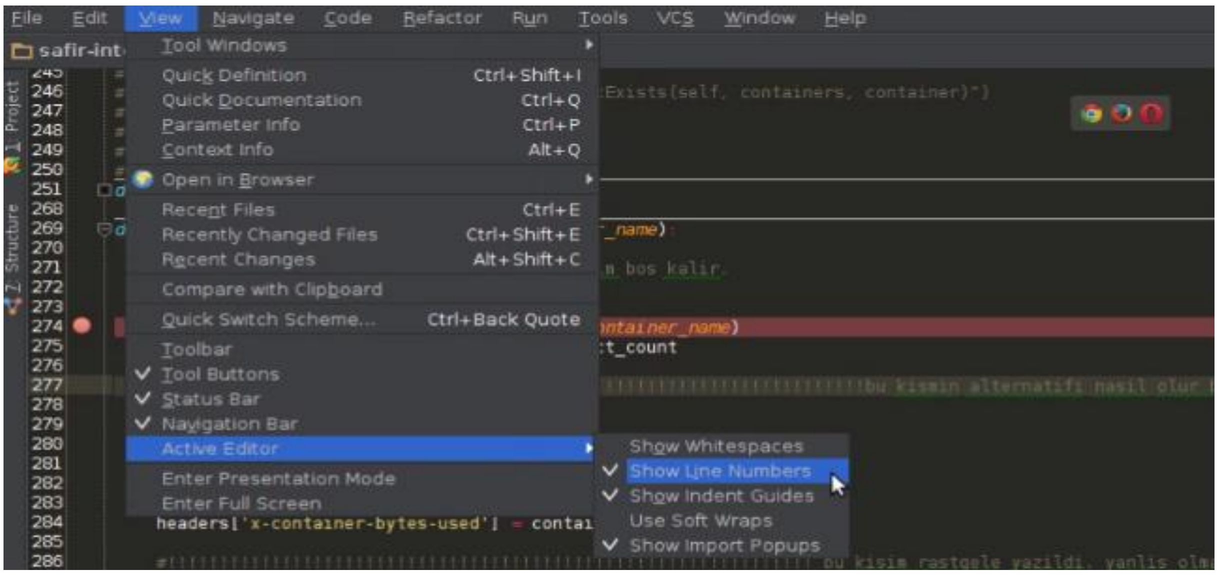


Figure 26. Line numbers

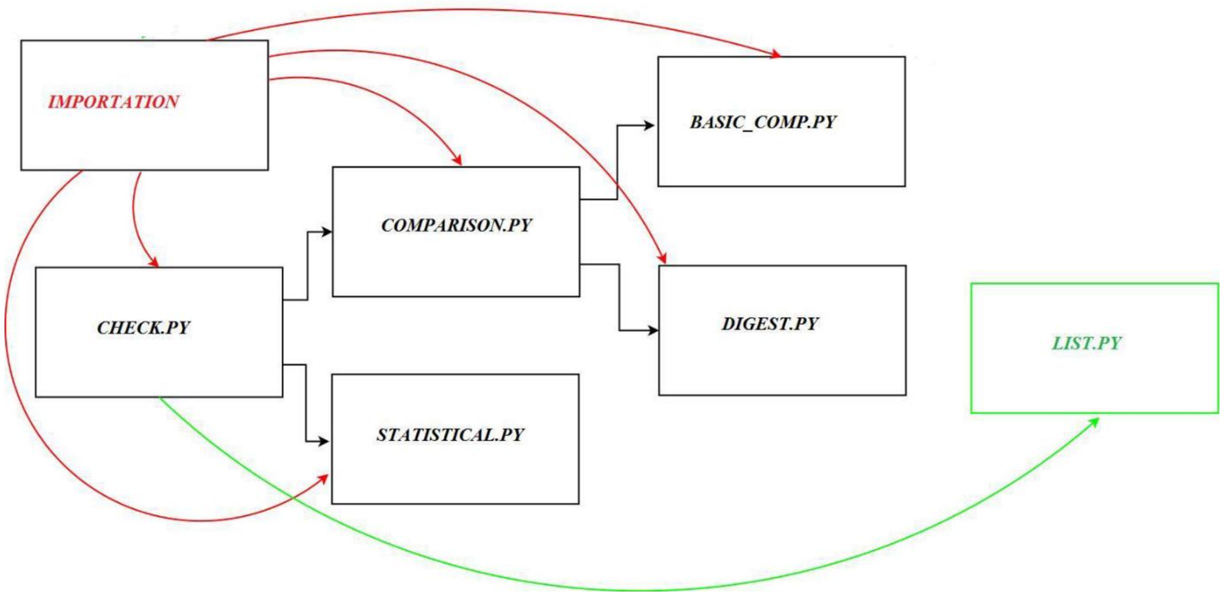
The last figure shows the easiest way to enable line numbers, something that is indeed practical. After all tools needed for the project installed, the development part can start.

3.2 Implementation

In this chapter, the development of the project will take place. Through the implementation phase all the code is written in Python, developing the tool to provide data transfer’s supervision. The descriptions below are explaining analytically, the ways that the project is developed. Generally, Python documentation includes many modules which are providing data comparison such as diff, dcmp, filecmp, difflip and so on. In this project, those modules are not used and they are not part of the program but therefore are used in the testing phase. The forenamed modules constitute a very useful tool which can be compared with the developed program and give results for the validity and clarity as well as the efficiency and the credibility of the project.

3.2.1 Scripts for comparison

To begin with, in python every single script can be imported by another script. Scripts are programs written for a special run-time environment that automate the execution of tasks that could alternatively be executed one-by-one by a human operator or



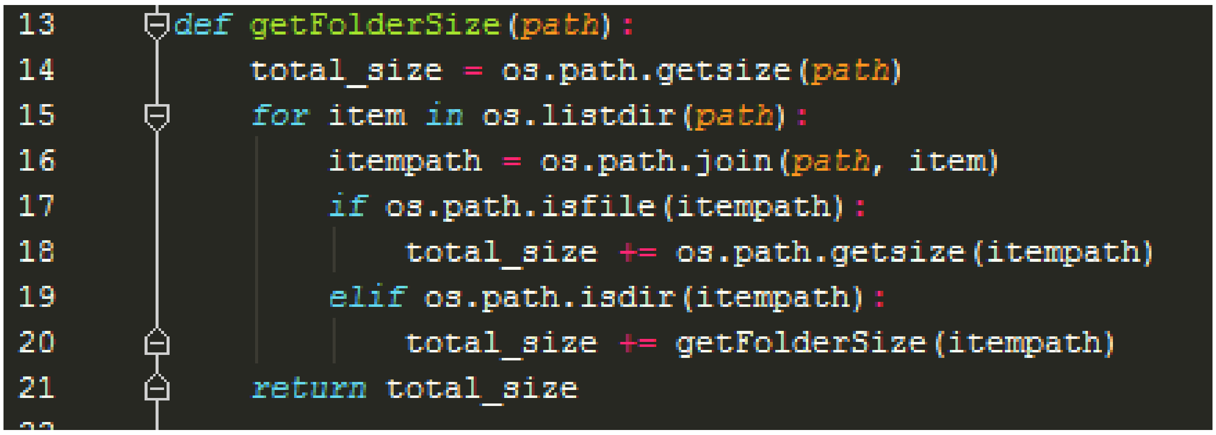
imported by each other. That allows the creation of clear and flexible code that does not need to be written in one script. In addition, it provides the capability to handle all the project from only one script that will bear the responsibility of running the other scripts properly. Below at figure 27 is displayed the way that the scripts are imported by others particularly for this project.

Figure 27. Scripts architecture

Apart the scripts depicted, there are few more scripts that will be imported and explained afterwards. Between others, there is the importation script which contains all the libraries used in the program and also the list.py which includes the lists of the arguments necessary for the comparison. (lists of folders paths) The user needs to put the paths of the folders which he desires to compare into the lists of the script called list.py. Once paths filled, they are automatically imported to the other scripts. To run the comparison is required only the script check.py. Once the check script runs then the comparison script and statistical one run in parallel. Moreover, the comparison script calls two other scripts named basic comparison and digest. The .py is a script file format used by Python and is similar with the popular .txt, .pdf, .doc extensions.

The supervision of data transfer is based mostly on two scripts, the statistic script and the comparison one. Let's focus on those two, the first one is created to provide general information about the folders under comparison, information such as the file size, the number of items, the number of folders and so on. The second one named comparison script is divided into two other scripts, the basic_comparison and the digest one. The reason that the statistics and the comparison are two different scripts is obvious as they don't do the same thing. Also, it is preferable to have those two scripts separate as they can be used independently. The fact that the comparison script is divided into two is because of technical reasons.

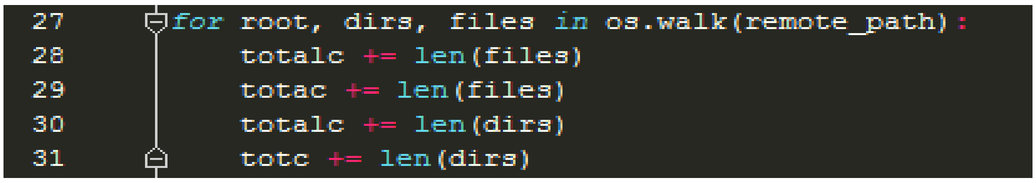
Below there are parts of the statistic script and its results. The whole script can be found in the appendices.



```
def getFolderSize(path):
    total_size = os.path.getsize(path)
    for item in os.listdir(path):
        itempath = os.path.join(path, item)
        if os.path.isfile(itempath):
            total_size += os.path.getsize(itempath)
        elif os.path.isdir(itempath):
            total_size += getFolderSize(itempath)
    return total_size
```

Figure 28. Statistic.py - folder size

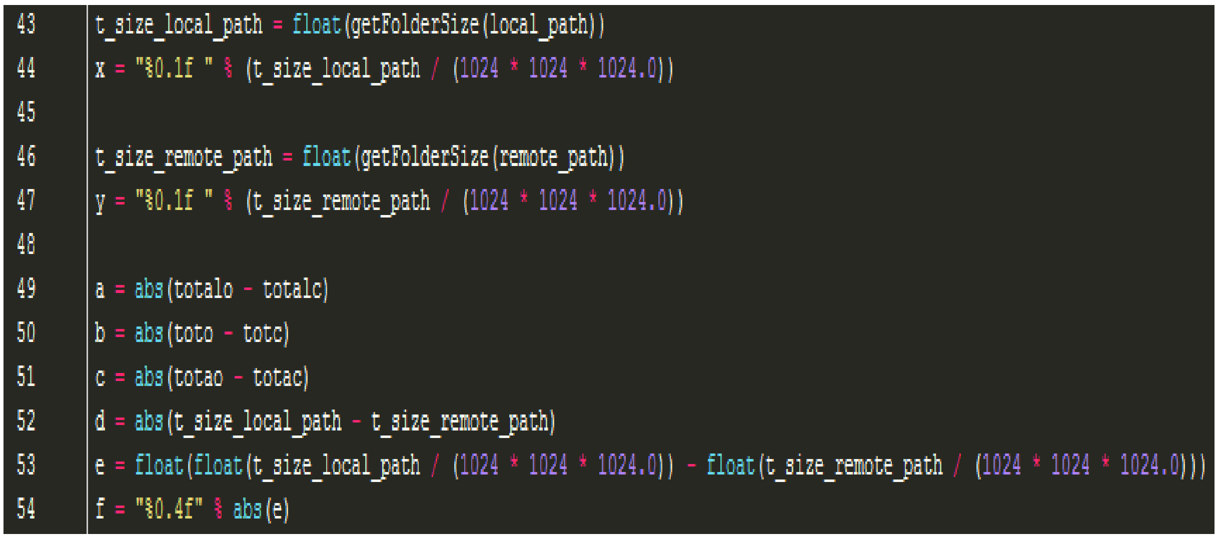
In figure 28 is shown the code which collects and provides basic information about the folders under comparison, such as the folder size. With the help of os.listdir() and os.path() functions, the program walks into directories and takes back the real folder size.



```
for root, dirs, files in os.walk(remote_path):
        totalc += len(files)
        totac += len(files)
        totalc += len(dirs)
        totc += len(dirs)
```

Figure 29. Statistic script-number of folders

In a similar way os.walk()[1] function walks into the directories while there is a loop and gives back the total number of items, folders and files. There are few more

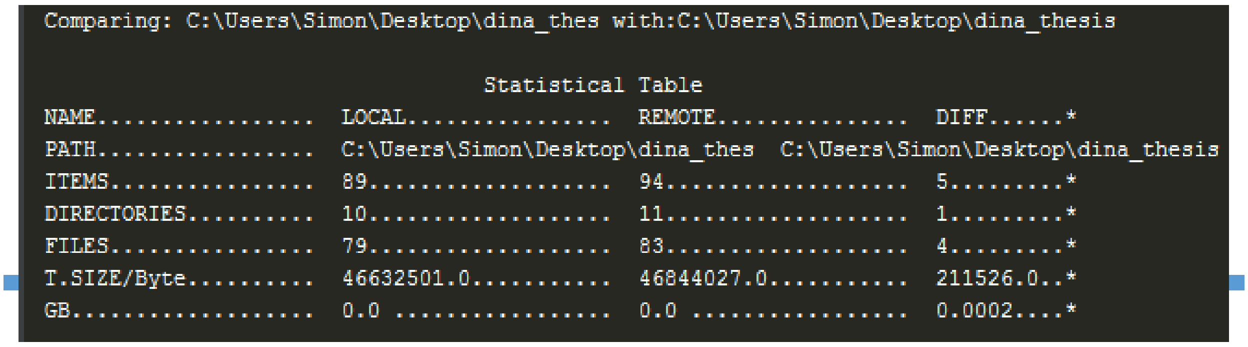


```
t_size_local_path = float(getFolderSize(local_path))
x = "%0.1f " % (t_size_local_path / (1024 * 1024 * 1024.0))

t_size_remote_path = float(getFolderSize(remote_path))
y = "%0.1f " % (t_size_remote_path / (1024 * 1024 * 1024.0))

a = abs(totalo - totalc)
b = abs(toto - totc)
c = abs(totao - totac)
d = abs(t_size_local_path - t_size_remote_path)
e = float(float(t_size_local_path / (1024 * 1024 * 1024.0)) - float(t_size_remote_path / (1024 * 1024 * 1024.0)))
f = "%0.4f" % abs(e)
```

functions used for statistical information which at the end will be displayed in a table. In the figure below the function abs () is used to return the absolute value of a number. The argument may be a plain, a long integer or a floating point number.

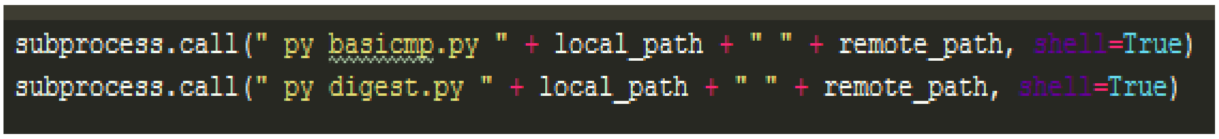
Figure 30. Statistic.py- table code

The result returned by the statistic script in the terminal looks like below:



```
Comparing: C:\Users\Simon\Desktop\dina_thes with:C:\Users\Simon\Desktop\dina_thesis

                              Statistical Table
NAME................  LOCAL...............  REMOTE..............  DIFF......*
PATH................  C:\Users\Simon\Desktop\dina_thes  C:\Users\Simon\Desktop\dina_thesis
ITEMS...............  89..................  94..................  5.........*
DIRECTORIES.........  10..................  11..................  1.........*
FILES...............  79..................  83..................  4.........*
T.SIZE/Byte.........  46632501.0..........  46844027.0..........  211526.0..*
GB..................  0.0 ................  0.0 ................  0.0002....*
```

Figure 31.statistic table-terminal

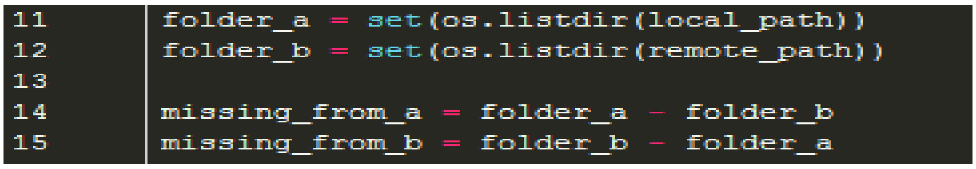


```
subprocess.call(" py basicmp.py " + local_path + " " + remote_path, shell=True)
subprocess.call(" py digest.py " + local_path + " " + remote_path, shell=True)
```

Figure 32.comparsion.py- call

The comparison script calls two other scripts, like below:

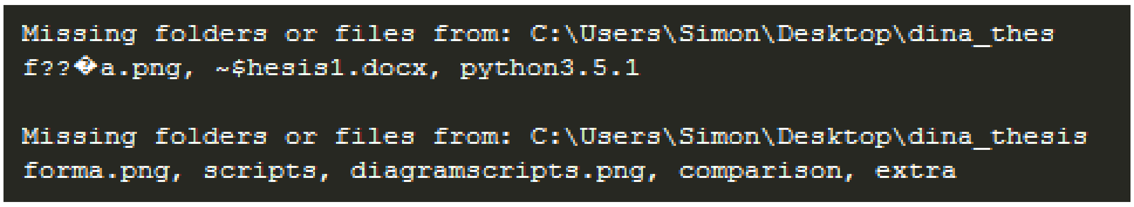
The comparison.py script is able to call other scripts because of the imported module sub process. The basic comparison script is responsible for walking into the desired folders, compare them and give back a list of missing folders or files from each folder compared to the other.



```
folder_a = set(os.listdir(local_path))
folder_b = set(os.listdir(remote_path))

missing_from_a = folder_a - folder_b
missing_from_b = folder_b - folder_a
```

Figure 33. Basicmp.py

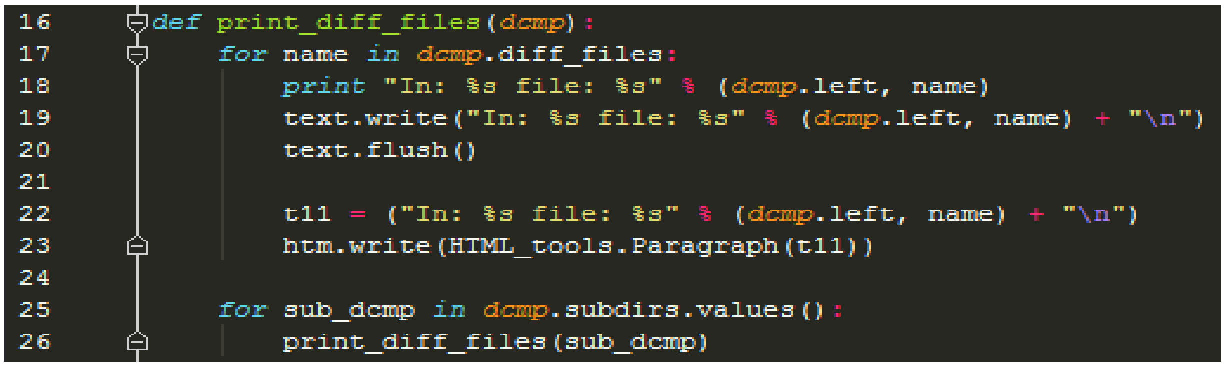
Basic comparison’s results are shown in the following picture. As you can see it detects missing all the missing files and folders even if they are hidden:



```
Missing folders or files from: C:\Users\Simon\Desktop\dina_thes
f??�a.png, ~$hesis1.docx, python3.5.1

Missing folders or files from: C:\Users\Simon\Desktop\dina_thesis
forma.png, scripts, diagramscripts.png, comparison, extra
```

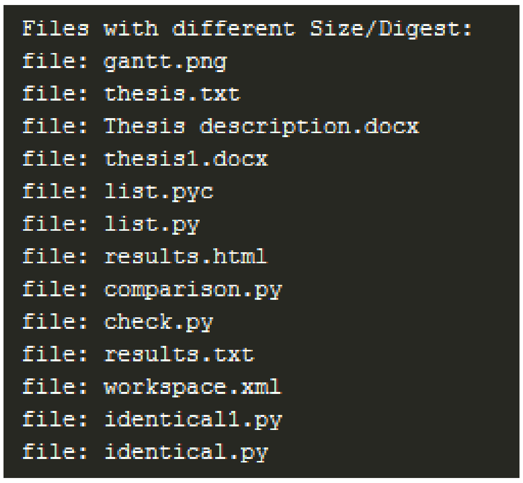
Figure 34. Basicmp.py-results

Part of the comparison is also the script called digest. The digest script is using the filecmp module as below, it detects the common files of the folders which having different size. The whole code for the dasicmp.py and digest.py can be found in the appendices.



```
def print_diff_files(dcmp):
    for name in dcmp.diff_files:
        print "In: %s file: %s" % (dcmp.left, name)
        text.write("In: %s file: %s" % (dcmp.left, name) + "\n")
        text.flush()

        t11 = ("In: %s file: %s" % (dcmp.left, name) + "\n")
        htm.write(HTML_tools.Paragraph(t11))

    for sub_dcmp in dcmp.subdirs.values():
        print_diff_files(sub_dcmp)
```

Figure 35. Comparison.py-digest.py

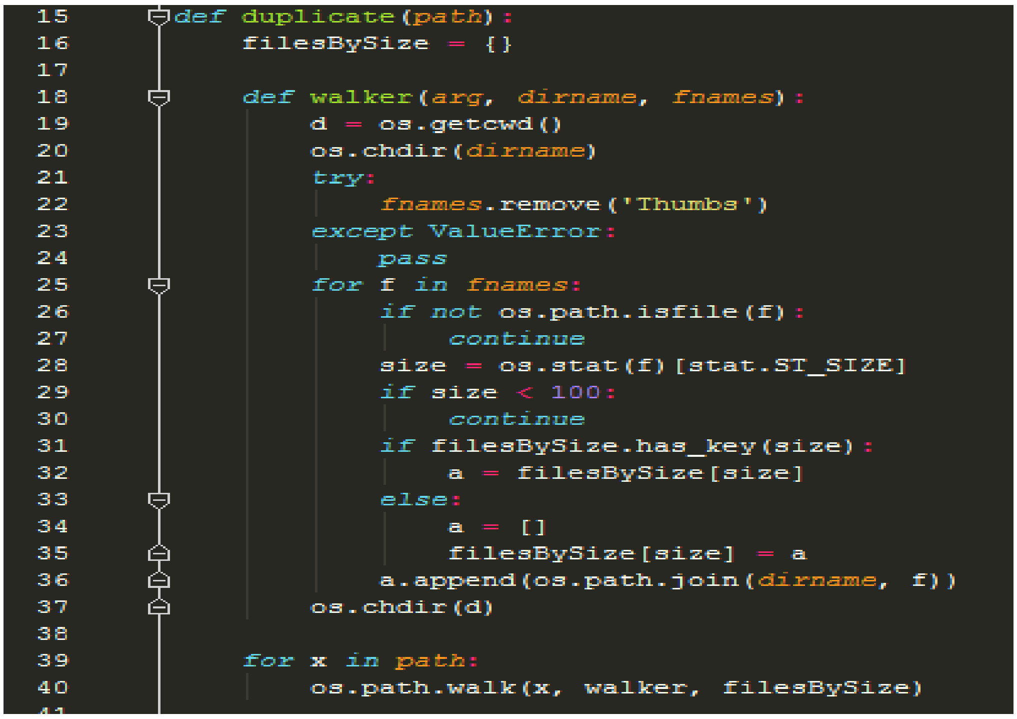


```
Files with different Size/Digest:
file: gantt.png
file: thesis.txt
file: Thesis description.docx
file: thesis1.docx
file: list.pyc
file: list.py
file: results.html
file: comparison.py
file: check.py
file: results.txt
file: workspace.xml
file: identical1.py
file: identical.py
```

Figure 36.comparison.py-digest re-

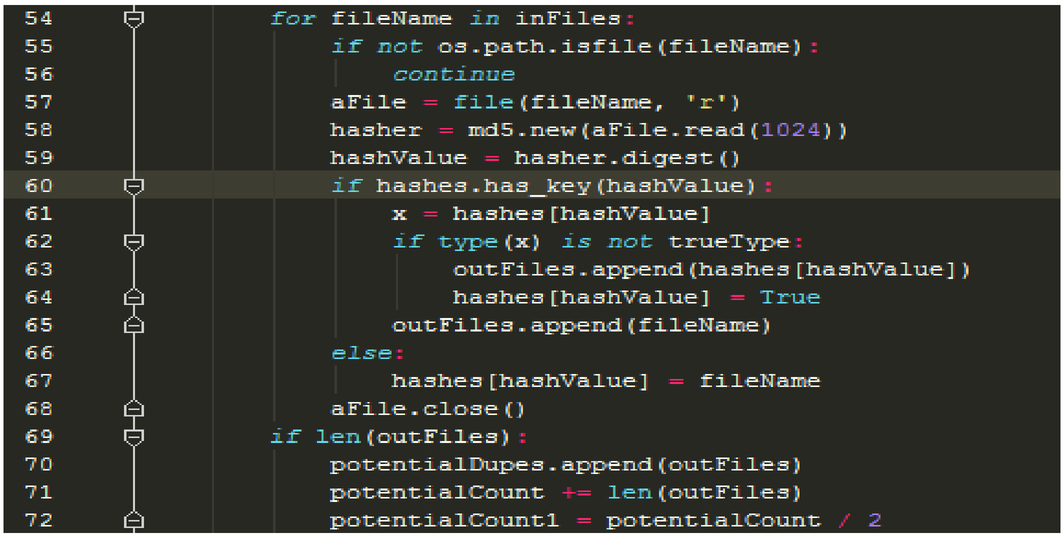
The result of the digest script is displayed on the right. The program is able to detect common files in directories and find out digest differences. Many times two or more files seem identical but they have some differences. For instance, the first file detected, called gantt.png is a picture in which is added a dot. The program detects any difference even though is just a dot.

To finish with the comparison scripts, last but not least, is the duplicate.py script. This one is not about the source and destination folders but it is about comparing files within the source or destination folder individually. As the name betrays, the duplicate script is looking for duplicated files. Using it in each folder may be useful because it happens that in the same folder there are unnecessary duplicates of a file. Parts of the code are underneath.



```python
def duplicate(path):
    filesBySize = {}

    def walker(arg, dirname, fnames):
        d = os.getcwd()
        os.chdir(dirname)
        try:
            fnames.remove('Thumbs')
        except ValueError:
            pass
        for f in fnames:
            if not os.path.isfile(f):
                continue
            size = os.stat(f)[stat.ST_SIZE]
            if size < 100:
                continue
            if filesBySize.has_key(size):
                a = filesBySize[size]
            else:
                a = []
                filesBySize[size] = a
            a.append(os.path.join(dirname, f))
        os.chdir(d)

    for x in path:
        os.path.walk(x, walker, filesBySize)
```

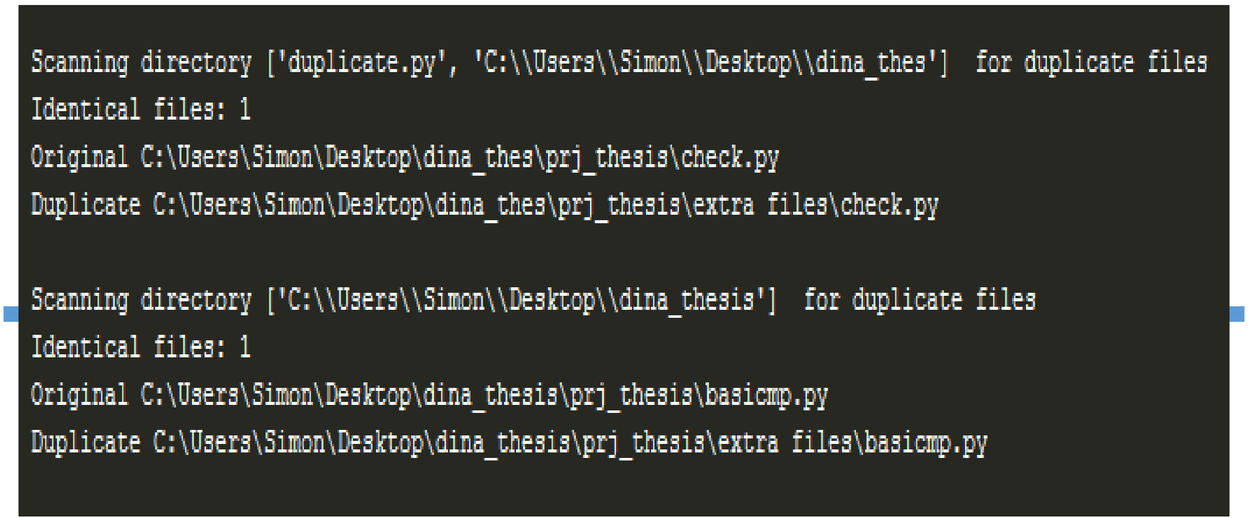
Figure 37. Duplicate.py -1



```python
        for fileName in inFiles:
            if not os.path.isfile(fileName):
                continue
            aFile = file(fileName, 'r')
            hasher = md5.new(aFile.read(1024))
            hashValue = hasher.digest()
            if hashes.has_key(hashValue):
                x = hashes[hashValue]
                if type(x) is not trueType:
                    outFiles.append(hashes[hashValue])
                    hashes[hashValue] = True
                outFiles.append(fileName)
            else:
                hashes[hashValue] = fileName
            aFile.close()
        if len(outFiles):
            potentialDupes.append(outFiles)
            potentialCount += len(outFiles)
            potentialCount1 = potentialCount / 2
```

Figure 38. Duplicate.py-2

The script is comparing the size of the files (bytes) and when two files have the



```
Scanning directory ['duplicate.py', 'C:\\Users\\Simon\\Desktop\\dina_thes']  for duplicate files
Identical files: 1
Original C:\Users\Simon\Desktop\dina_thes\prj_thesis\check.py
Duplicate C:\Users\Simon\Desktop\dina_thes\prj_thesis\extra files\check.py

Scanning directory ['C:\\Users\\Simon\\Desktop\\dina_thesis']  for duplicate files
Identical files: 1
Original C:\Users\Simon\Desktop\dina_thesis\prj_thesis\basicmp.py
Duplicate C:\Users\Simon\Desktop\dina_thesis\prj_thesis\extra files\basicmp.py
```

same size then the script compares the hash of each file, with the help of the MD5 algorithm. The MD5 message digest algorithm is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32 digit hexadecimal number. MD5 has been utilized in a wide variety of cryptographic applications and is also commonly used to verify data integrity. Point this script in a folder or several folders and it will find all duplicate files within the folders, referring the source file as the original one in case of duplicates. It is designed to handle hundreds of thousands of files of any size at a time and to do it very quickly. The results are given from the script look like as below:

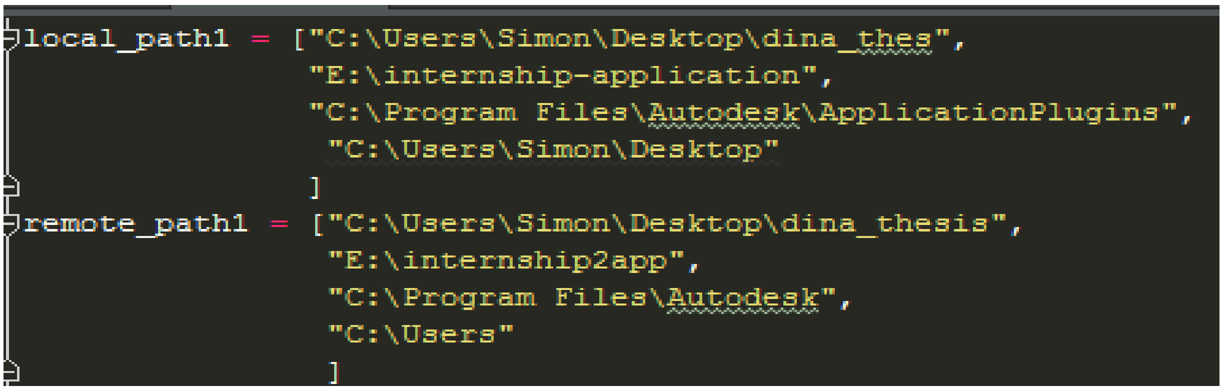
To conclude with, after presenting all the scripts which are participating in the comparison process, here is a summary:

- Statistic.py, gives all the statistical information about the compared directories (source/destination).
- Comparison.py, calls two other scripts to be used.
    - Basic comparison.py, detects the missing folders and files.
    - Digest.py, detects the size differences between similar files.
- Duplicate.py which compares same size files and finds out the identical.

Figure 39. Duplicate.py - results

3.2.2 Other scripts

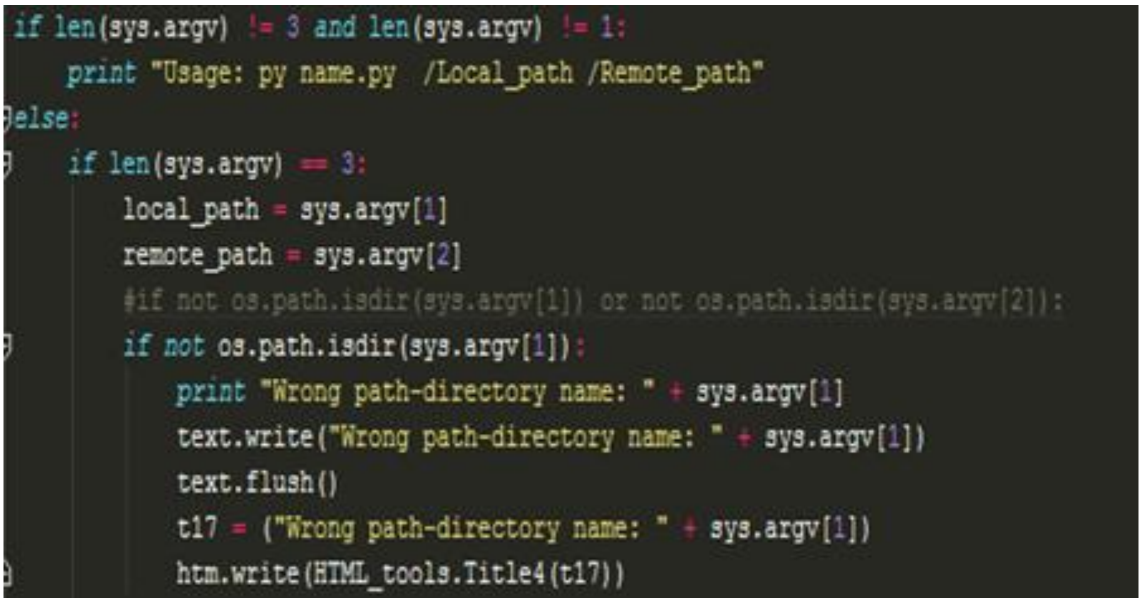
Here will be explained the rest of the project apart from the comparison phase previosuly described. The idea of the project is to supervise data transfers without size limitation and without limitation of folders per each time. To be clear, the project includes a script list where the user is able to put the folders paths.



```
local_path1 = ["C:\Users\Simon\Desktop\dina_thes",
                "E:\internship-application",
                "C:\Program Files\Autodesk\ApplicationPlugins",
                 "C:\Users\Simon\Desktop"
                ]
remote_path1 = ["C:\Users\Simon\Desktop\dina_thesis",
                 "E:\internship2app",
                 "C:\Program Files\Autodesk",
                 "C:\Users"
                 ]
```

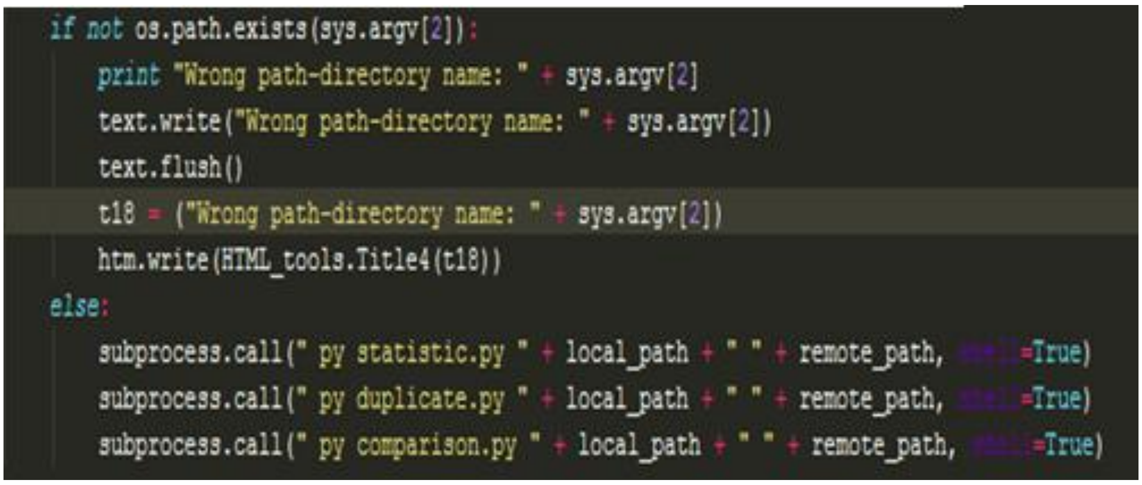
Figure 40. List.py

Each time the local path1 compares with the remote path1, source and destination, so the user should put all folders paths he/she wishes to compare in the local path1 list and then all folders paths, that needs to be compare with, into the remote path1 list. In order for this to happen, it is necessary that the check script take action. The check script contains a loop which allows the program to run continuously all paths and needs just one execution in order to compare the numerous folders.



```
if len(sys.argv) != 3 and len(sys.argv) != 1:
    print "Usage: py name.py  /Local_path /Remote_path"
else:
    if len(sys.argv) == 3:
        local_path = sys.argv[1]
        remote_path = sys.argv[2]
        #if not os.path.isdir(sys.argv[1]) or not os.path.isdir(sys.argv[2]):
        if not os.path.isdir(sys.argv[1]):
            print "Wrong path-directory name: " + sys.argv[1]
            text.write("Wrong path-directory name: " + sys.argv[1])
            text.flush()
            t17 = ("Wrong path-directory name: " + sys.argv[1])
            htm.write(HTML_tools.Title4(t17))
```

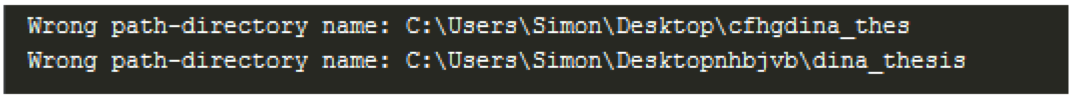
Figure 41. Check.py



```
if not os.path.exists(sys.argv[2]):
    print "Wrong path-directory name: " + sys.argv[2]
    text.write("Wrong path-directory name: " + sys.argv[2])
    text.flush()
    t18 = ("Wrong path-directory name: " + sys.argv[2])
    htm.write(HTML_tools.Title4(t18))
else:
    subprocess.call(" py statistic.py " + local_path + " " + remote_path, [illegible]=True)
    subprocess.call(" py duplicate.py " + local_path + " " + remote_path, [illegible]=True)
    subprocess.call(" py comparison.py " + local_path + " " + remote_path, [illegible]=True)
```

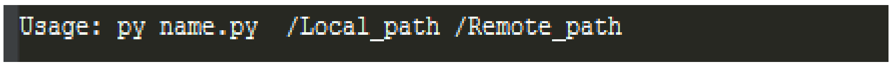
Figure 42.Check.py2

Check.py includes all the possible scenarios, such as paths that do not exist or if the user gave fewer arguments than the necessary and so on. In addition, it allows the user to choose if he wants to run a script separately, for instance if he needs to check only for duplicates files, and also he can do the comparison only on a pair of folders through the terminal or for plenty of them by using the list.



```
Wrong path-directory name: C:\Users\Simon\Desktop\cfhgdina_thes
Wrong path-directory name: C:\Users\Simon\Desktopnhbjvb\dina_thesis
```

Figure 43.Check.py- results



```
Usage: py name.py  /Local_path /Remote_path
```

Figure 44. Check.py usage

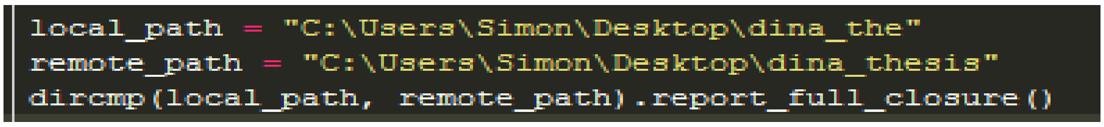
3.2 Test phase

Another important phase, as important as the implementation one, is the test phase. During this phase, the program can be tested in many ways in order to improve it. Many factors influence the performance of the project and they need to be identified and faced. To begin with, the test phase started with testing the program by changing the data and performing different test scenarios:

- ❖ Replace a letter in a file, in order to keep the same size, but the digest to be different.
- ❖ Create hidden files and add them in the folders.

- ❖ Include in existing data many different formats, such as .pdf, .doc, .xml, .py, .exe, etc. Actually, all the type of data which can be found in a usual computer.
- ❖ Duplicate folders in different folders
- ❖ Duplicate files in different folders
- ❖ Delete folders/files
- ❖ Add images /modify images/ add a dot on them
- ❖ Add Videos /modify videos.
- ❖ Add symbolic links to see if they can be followed.

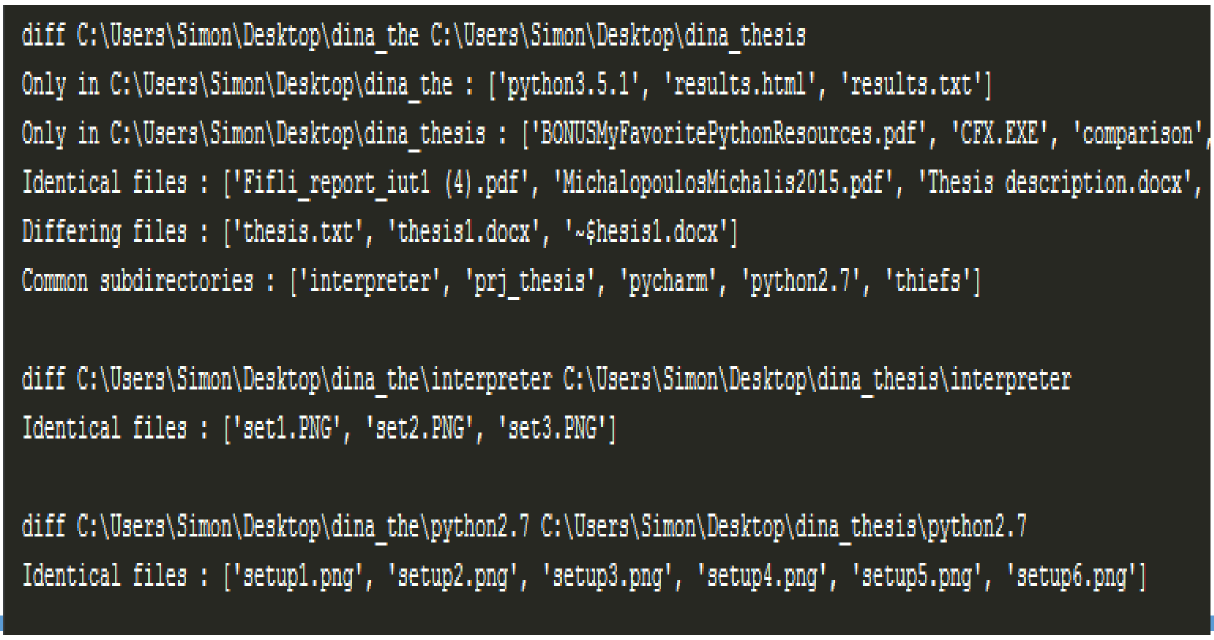
After modifying data and comparing them once more, the expected results were returned. To evaluate the quality of the program next step was to compare those results with results provided by a standard and reliable tool called dircmp, which is a free Python module available in its documentation. (*https://docs.python.org/2/library/filecmp.html*)



```
local_path = "C:\Users\Simon\Desktop\dina_the"
remote_path = "C:\Users\Simon\Desktop\dina_thesis"
dircmp(local_path, remote_path).report_full_closure()
```

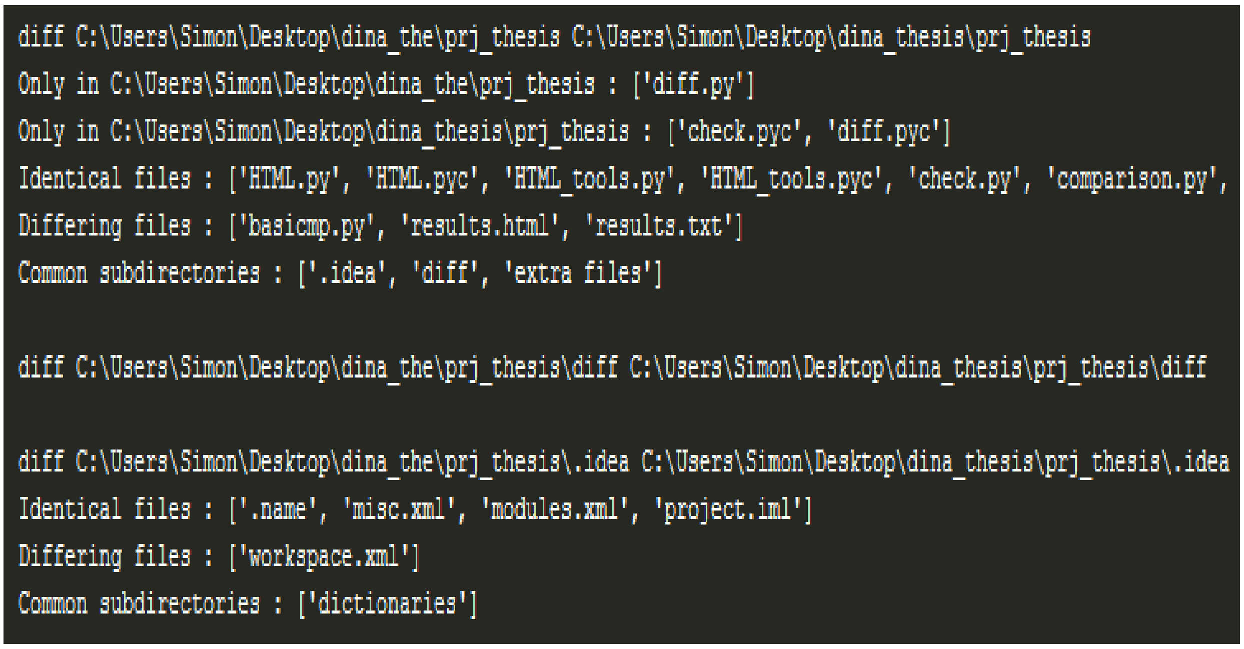
Figure 45. Dircmp

The above capture displays a dircmp method comparing directories, a module that is available in Python's standard documentation. The dircmp class compares files by doing shallow comparisons and the report_full_closure() prints a comparison between a and b and common subdirectories recursively.



```
diff C:\Users\Simon\Desktop\dina_the C:\Users\Simon\Desktop\dina_thesis
Only in C:\Users\Simon\Desktop\dina_the : ['python3.5.1', 'results.html', 'results.txt']
Only in C:\Users\Simon\Desktop\dina_thesis : ['BONUSMyFavoritePythonResources.pdf', 'CFX.EXE', 'comparison',
Identical files : ['Fifli_report_iut1 (4).pdf', 'MichalopoulosMichalis2015.pdf', 'Thesis description.docx',
Differing files : ['thesis.txt', 'thesis1.docx', '~$hesis1.docx']
Common subdirectories : ['interpreter', 'prj_thesis', 'pycharm', 'python2.7', 'thiefs']

diff C:\Users\Simon\Desktop\dina_the\interpreter C:\Users\Simon\Desktop\dina_thesis\interpreter
Identical files : ['set1.PNG', 'set2.PNG', 'set3.PNG']

diff C:\Users\Simon\Desktop\dina_the\python2.7 C:\Users\Simon\Desktop\dina_thesis\python2.7
Identical files : ['setup1.png', 'setup2.png', 'setup3.png', 'setup4.png', 'setup5.png', 'setup6.png']
```

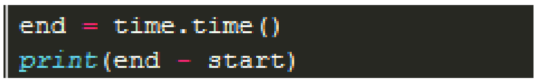
Figure 46. Dircmp- Results1



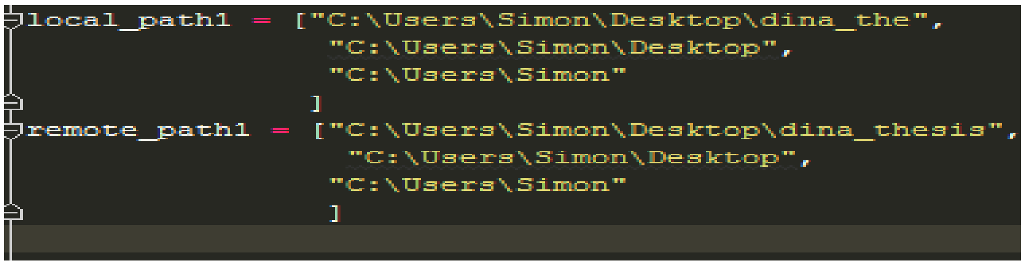
```
diff C:\Users\Simon\Desktop\dina_the\prj_thesis C:\Users\Simon\Desktop\dina_thesis\prj_thesis
Only in C:\Users\Simon\Desktop\dina_the\prj_thesis : ['diff.py']
Only in C:\Users\Simon\Desktop\dina_thesis\prj_thesis : ['check.pyc', 'diff.pyc']
Identical files : ['HTML.py', 'HTML.pyc', 'HTML_tools.py', 'HTML_tools.pyc', 'check.py', 'comparison.py',
Differing files : ['basicmp.py', 'results.html', 'results.txt']
Common subdirectories : ['.idea', 'diff', 'extra files']

diff C:\Users\Simon\Desktop\dina_the\prj_thesis\diff C:\Users\Simon\Desktop\dina_thesis\prj_thesis\diff

diff C:\Users\Simon\Desktop\dina_the\prj_thesis\.idea C:\Users\Simon\Desktop\dina_thesis\prj_thesis\.idea
Identical files : ['.name', 'misc.xml', 'modules.xml', 'project.iml']
Differing files : ['workspace.xml']
Common subdirectories : ['dictionaries']
```

Figure 47. Dircmp - Results2

On the figures 46 and 47 are captured the results the dircmp module. Obviously the dircmp comparison gives all the folders details but in a different way than the program of this thesis does. When paying attention at lists named “different files” and comparing them with the figures 34 and 36, the results which are coming out ensure that the program works as it is supposed to.

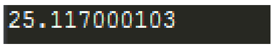
Next step is to test the program on numerous directories at the same time and by applying timers at the beginning and at the end of the check script in order to check how much time is needed for the execu-  tion.

```
end = time.time()
print(end - start)
```



```
local_path1 = ["C:\Users\Simon\Desktop\dina_the",
               "C:\Users\Simon\Desktop",
               "C:\Users\Simon"
              ]
remote_path1 = ["C:\Users\Simon\Desktop\dina_thesis",
                "C:\Users\Simon\Desktop",
               "C:\Users\Simon"
                ]
```

Figure 48 New List-timer-sec

For instance, the execution time required to compare 1 Gigabytes of data with more than 7000 items was  seconds.

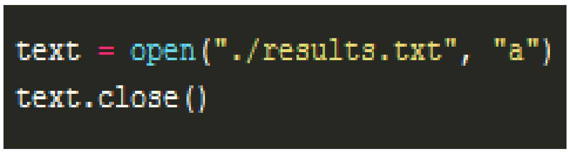
```
25.117000103
```

Chapter 4. Results

To summarize until now are described, the implementation and the test phase. During the implementation phase, the code was written in Python and developed through the IDE PyCharm. The results of this project are returned on a terminal. The results include a statistical table, the missing files from the compared paths, and duplicated files in the comparing directories. Also, the results include the ostensibly same files but with different digest. There is a need from the user to have the results exported in a file, an easy way to read and transfer them. Also can't the user have the ability to read the results within an HTML display? Something that is indeed quite common and practical.

4.1 Text report

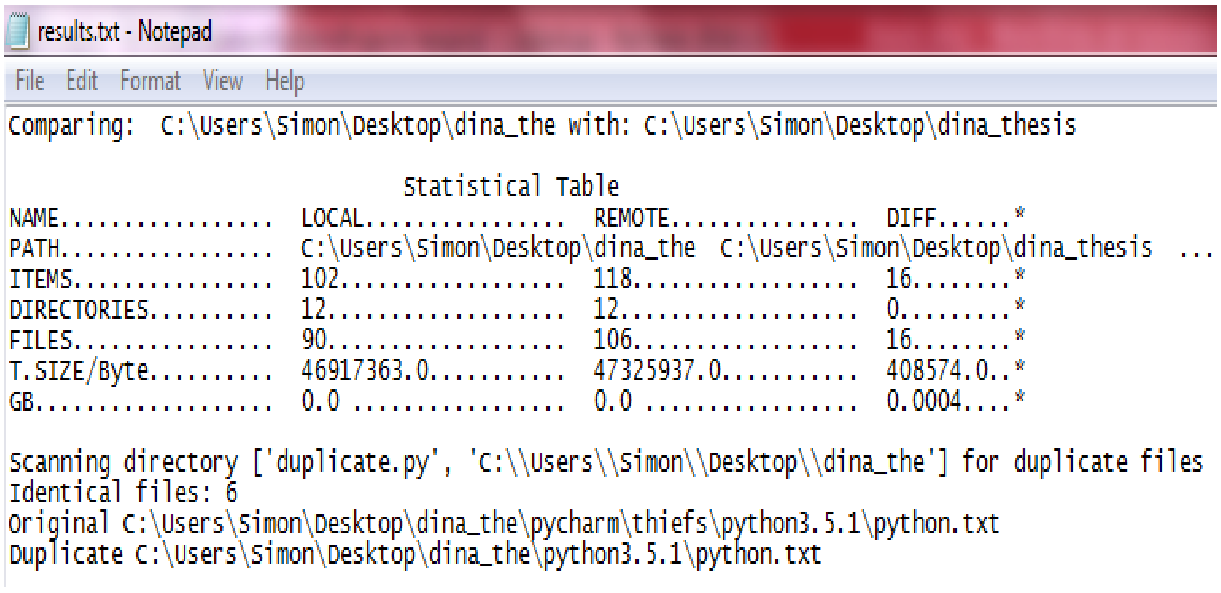
Between many file types, such as PDF, Word, Excel and txt the chosen format to display the results was a .txt file. It was selected because it requires fewer code lines comparing to the others. It's easy to implement it and is even easier for the users to read the results without having to download a third party tool or plugin (like Microsoft word etc.). In order to get the results written in a txt file, it is necessary that every script includes two code lines like below:



```
text = open("./results.txt", "a")
text.close()
```

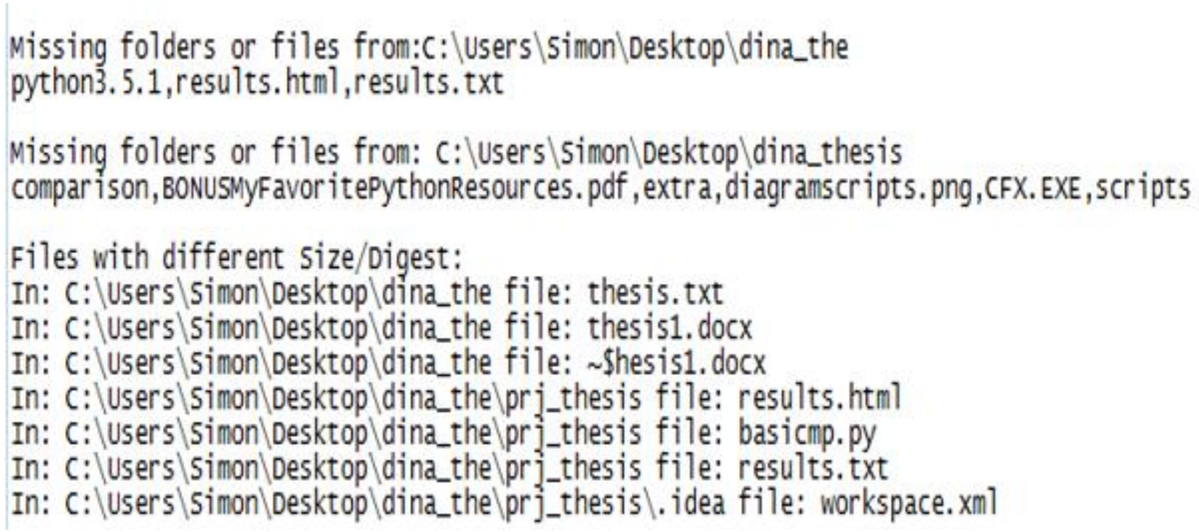
Figure 49. Txt command

Thanks to the first line a txt file is created named "results.txt" and it is going to be in the same file as the current program. So in this case the ". /" path is used but it could be any other path instead. The results will be exported in the txt file as below.



```
Comparing:  C:\Users\Simon\Desktop\dina_the with: C:\Users\Simon\Desktop\dina_thesis

                              Statistical Table
NAME................  LOCAL...............  REMOTE..............  DIFF......*
PATH................  C:\Users\Simon\Desktop\dina_the  C:\Users\Simon\Desktop\dina_thesis  ...
ITEMS...............  102.................  118.................  16........*
DIRECTORIES.........  12..................  12..................  0.........*
FILES...............  90..................  106.................  16........*
T.SIZE/Byte.........  46917363.0..........  47325937.0..........  408574.0..*
GB..................  0.0 ................  0.0 ................  0.0004....*

Scanning directory ['duplicate.py', 'C:\\Users\\Simon\\Desktop\\dina_the'] for duplicate files
Identical files: 6
Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\python.txt
Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\python.txt
```

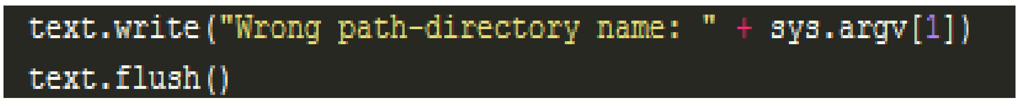
Figure 50.Results1- TXT



```
Missing folders or files from:C:\Users\Simon\Desktop\dina_the
python3.5.1,results.html,results.txt

Missing folders or files from: C:\Users\Simon\Desktop\dina_thesis
comparison,BONUSMyFavoritePythonResources.pdf,extra,diagramscripts.png,CFX.EXE,scripts

Files with different Size/Digest:
In: C:\Users\Simon\Desktop\dina_the file: thesis.txt
In: C:\Users\Simon\Desktop\dina_the file: thesis1.docx
In: C:\Users\Simon\Desktop\dina_the file: ~$hesis1.docx
In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.html
In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: basicmp.py
In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.txt
In: C:\Users\Simon\Desktop\dina_the\prj_thesis\.idea file: workspace.xml
```

Figure 51. Results2 - TXT

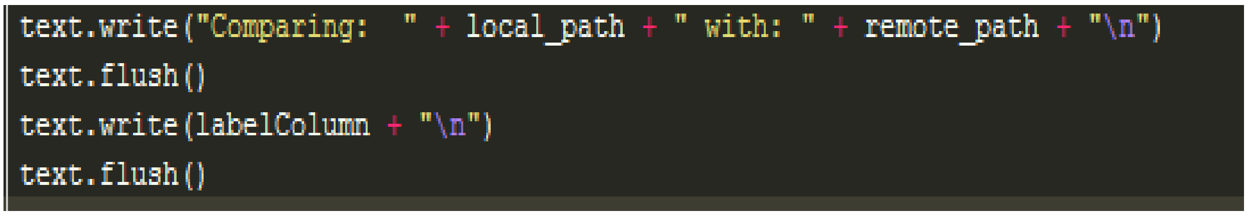
The above results are coming from the combination of the following code parts. Indeed, each script has his own “export-to-txt” code part.



```
text.write("Wrong path-directory name: " + sys.argv[1])
text.flush()
```

Figure 52. Check.py –TXT

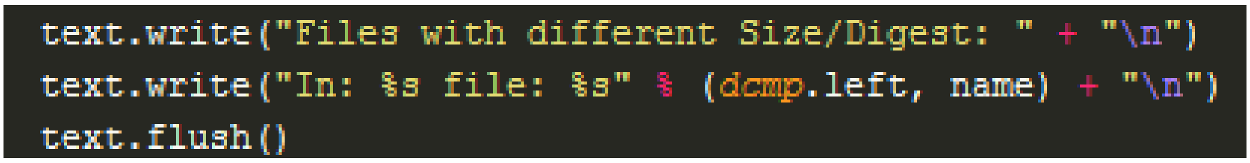
Between the parentheses contain the desired text to be displayed. For example here a warning will be returned to the user if the path that he/she gave was wrong. The sys.argv[1] is pointing to the argument number, in this example the first path inserted by user was wrong.



```
text.write("Comparing:  " + local_path + " with: " + remote_path + "\n")
text.flush()
text.write(labelColumn + "\n")
text.flush()
```

Figure 53. Statistic.py- TXT

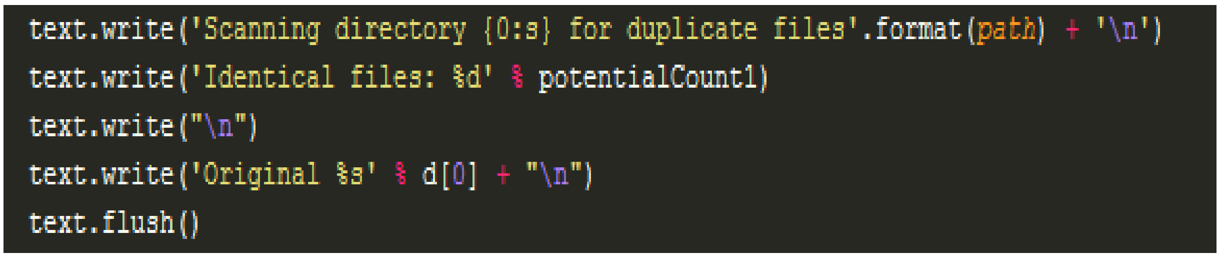
Same here, parenthesis contain the desired text displayed and the “+” sign allows to concatenate variables to it. In this example the local and remote paths variables are used.



```
text.write("Files with different Size/Digest: " + "\n")
text.write("In: %s file: %s" % (dcmp.left, name) + "\n")
text.flush()
```

Figure 54. Digest.py –TXT

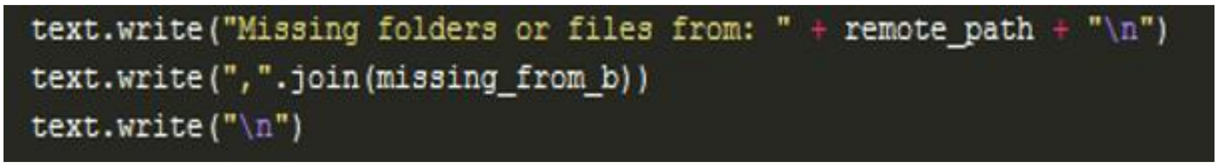
Depending on variables, it is necessary to declare them differently. For instance as %s for a string, as %d for an integer and %f for a float.



```
text.write('Scanning directory {0:s} for duplicate files'.format(path) + '\n')
text.write('Identical files: %d' % potentialCount1)
text.write("\n")
text.write('Original %s' % d[0] + "\n")
text.flush()
```

Figure 55. Duplicate.py -TXT

Above, the .format(path) is responsible to print the current scanning paths of the directories and below the join.() is helping to print lists into a TXT file.

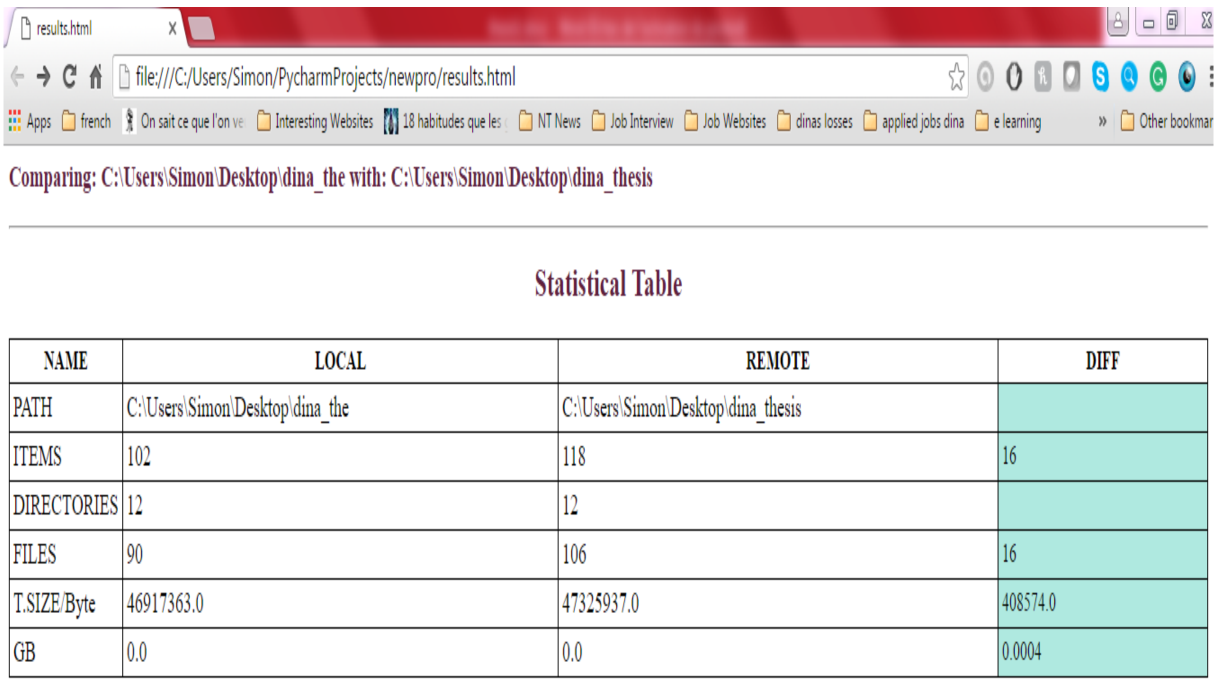


```
text.write("Missing folders or files from: " + remote_path + "\n")
text.write(",".join(missing_from_b))
text.write("\n")
```

Figure 56. Basicmp.py - TXT

4.2 HTML report

Generate a HTML report can be complicated or simple depending on how much complex the results are or how much complexed layouts are needed. In this project a simple and friendly HTML display is used because the project itself aims to facilitate the user's life. The results are well organized and easy to read. A sample of the results in HTML format is displayed below. More samples are available in the appendices.



Comparing: C:\Users\Simon\Desktop\dina_the with: C:\Users\Simon\Desktop\dina_thesis

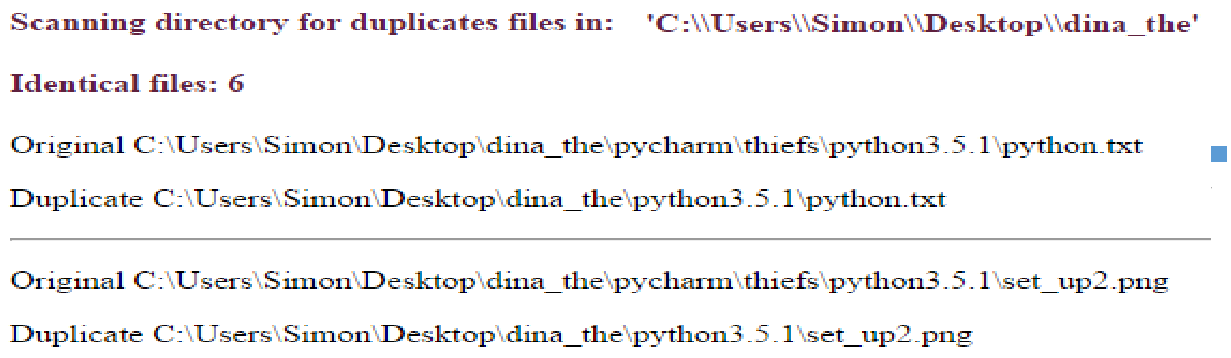
Statistical Table

NAME	LOCAL	REMOTE	DIFF
PATH	C:\Users\Simon\Desktop\dina_the	C:\Users\Simon\Desktop\dina_thesis	
ITEMS	102	118	16
DIRECTORIES	12	12	
FILES	90	106	16
T.SIZE/Byte	46917363.0	47325937.0	408574.0
GB	0.0	0.0	0.0004

Figure 57. HTML report- results1

On the above screenshot, you can see that results are visible via a web browser and they are organized in an HTML table with different colors and titles. The information are listed in columns and all the useful statistics are gathered and displayed within the HTML table.

The figure 58 shows the HTML report generated by the project and particularly by the statistics script. The following screenshot is also part of the report and is generated by the "duplicated" script. Titles in different colors are here to mention the results of duplicated files and their paths.



Scanning directory for duplicates files in: 'C:\\Users\\Simon\\Desktop\\dina_the'

Identical files: 6

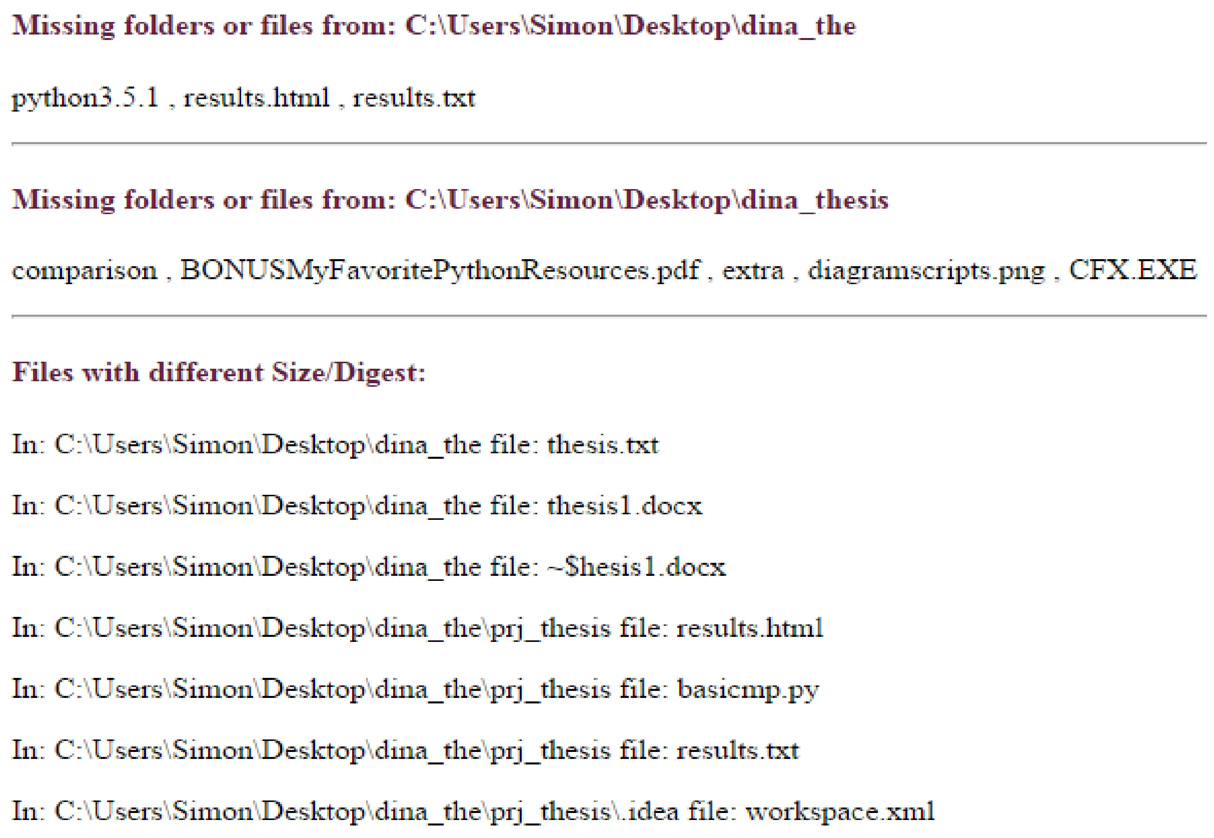
Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\python.txt

Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\python.txt

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\set_up2.png

Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\set_up2.png

Figure 58. HTML report- results2



Missing folders or files from: C:\Users\Simon\Desktop\dina_the

python3.5.1 , results.html , results.txt

Missing folders or files from: C:\Users\Simon\Desktop\dina_thesis

comparison , BONUSMyFavoritePythonResources.pdf , extra , diagramscripts.png , CFX.EXE

Files with different Size/Digest:

In: C:\Users\Simon\Desktop\dina_the file: thesis.txt

In: C:\Users\Simon\Desktop\dina_the file: thesis1.docx

In: C:\Users\Simon\Desktop\dina_the file: ~$hesis1.docx

In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.html

In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: basicmp.py

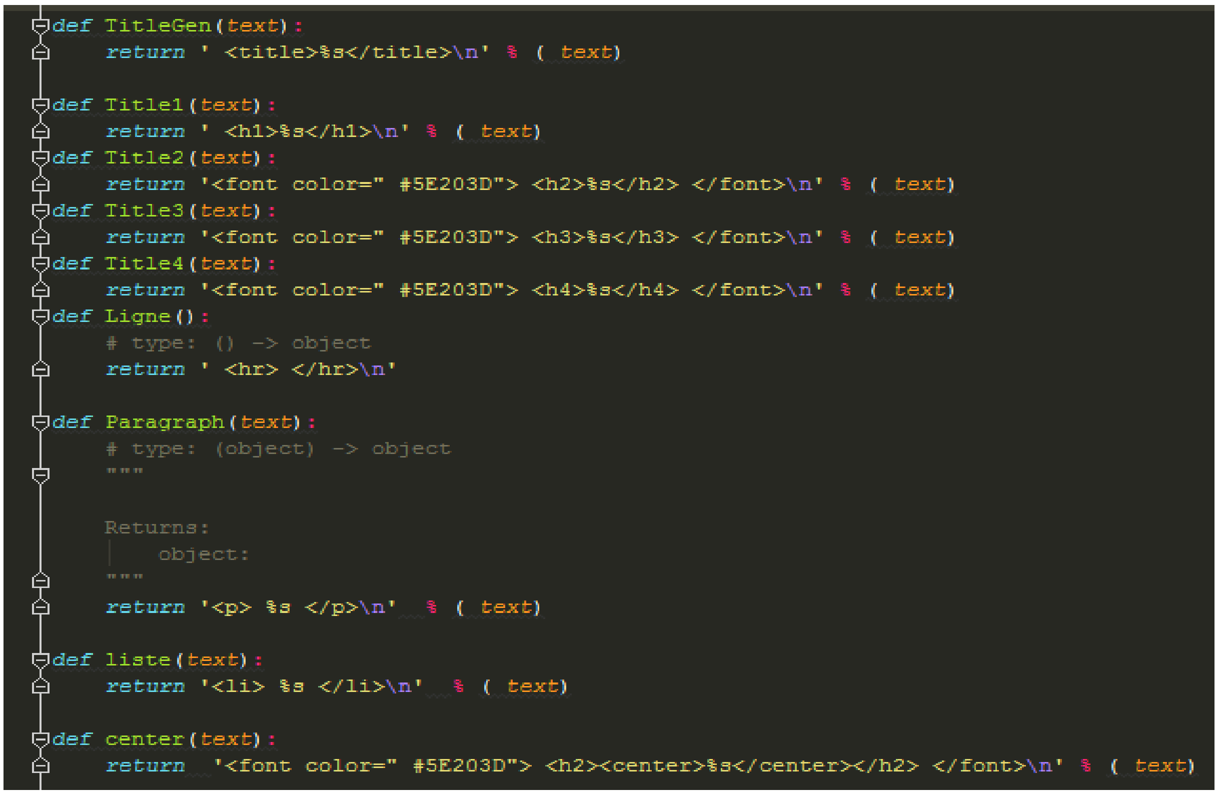
In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.txt

In: C:\Users\Simon\Desktop\dina_the\prj_thesis\.idea file: workspace.xml

Figure 59. HTML report- results3

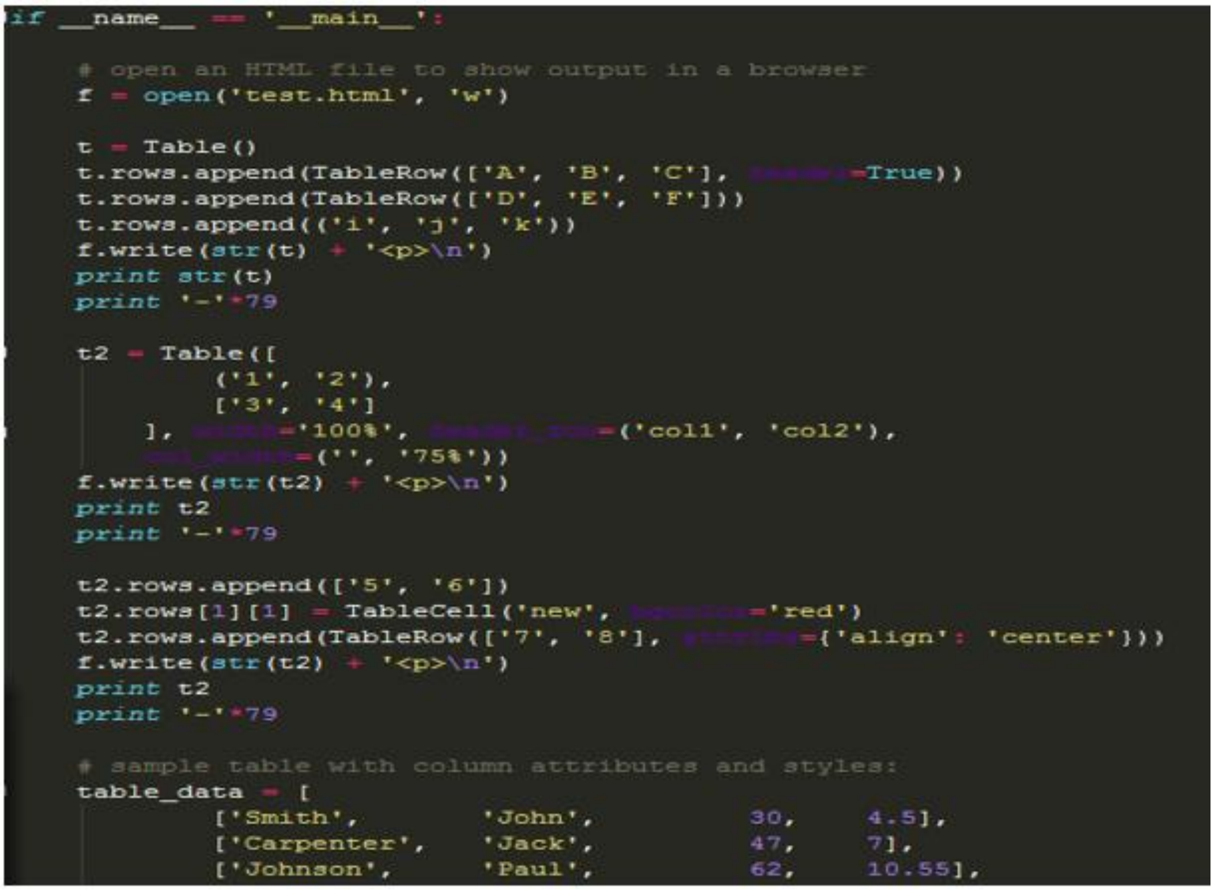
Above in the figures 59 and 60 are respectively captured the results of the “basic comparison” and “digest” scripts. By comparing the two directories, the program detects if there are missing folders from each directory compared to the other and gives the paths of the corresponding files. The last script is underlining files that seem identical but that are not.

The HTML report is created thanks to two scripts. Those scripts are imported in the project and they are respectively named HTML and HTML tools. The copyrights of the HTML.py belong to the cooperation between Philippe Lagadec (http://www.cecill.info) and the Institute Laue–Langevin. Some parts appear below and the whole scripts are included in the appendices.



```python
def TitleGen(text):
    return ' <title>%s</title>\n' % ( text)

def Title1(text):
    return ' <h1>%s</h1>\n' % ( text)
def Title2(text):
    return '<font color=" #5E203D"> <h2>%s</h2> </font>\n' % ( text)
def Title3(text):
    return '<font color=" #5E203D"> <h3>%s</h3> </font>\n' % ( text)
def Title4(text):
    return '<font color=" #5E203D"> <h4>%s</h4> </font>\n' % ( text)
def Ligne():
    # type: () -> object
    return ' <hr> </hr>\n'

def Paragraph(text):
    # type: (object) -> object
    """

    Returns:
        object:
    """
    return '<p> %s </p>\n'  % ( text)

def liste(text):
    return '<li> %s </li>\n'  % ( text)

def center(text):
    return  '<font color=" #5E203D"> <h2><center>%s</center></h2> </font>\n' % ( text)
```

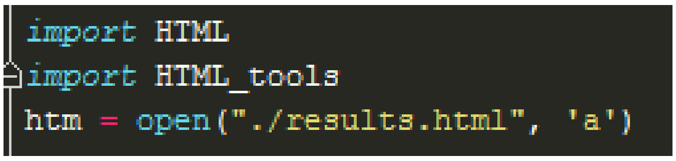
Figure 60. HTML_tools.py



```python
if __name__ == '__main__':

    # open an HTML file to show output in a browser
    f = open('test.html', 'w')

    t = Table()
    t.rows.append(TableRow(['A', 'B', 'C'], [illegible]=True))
    t.rows.append(TableRow(['D', 'E', 'F']))
    t.rows.append(('i', 'j', 'k'))
    f.write(str(t) + '<p>\n')
    print str(t)
    print '-'*79

    t2 = Table([
            ('1', '2'),
            ['3', '4']
        ], [illegible]='100%', [illegible]=('col1', 'col2'),
        [illegible]=('', '75%'))
    f.write(str(t2) + '<p>\n')
    print t2
    print '-'*79

    t2.rows.append(['5', '6'])
    t2.rows[1][1] = TableCell('new', [illegible]='red')
    t2.rows.append(TableRow(['7', '8'], [illegible]={'align': 'center'}))
    f.write(str(t2) + '<p>\n')
    print t2
    print '-'*79

    # sample table with column attributes and styles:
    table_data = [
            ['Smith',       'John',        30,    4.5],
            ['Carpenter',   'Jack',        47,    7],
            ['Johnson',     'Paul',        62,    10.55],
```

Figure 61 HTML.py

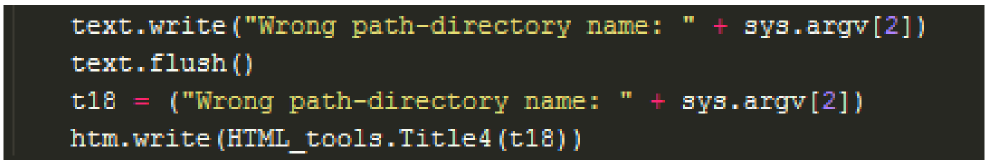
To generate an HTML report for this project, it is needed for each script of the program to include three lines as shown:



```
import HTML
import HTML_tools
htm = open("./results.html", 'a')
```

Figure 62.HTML code

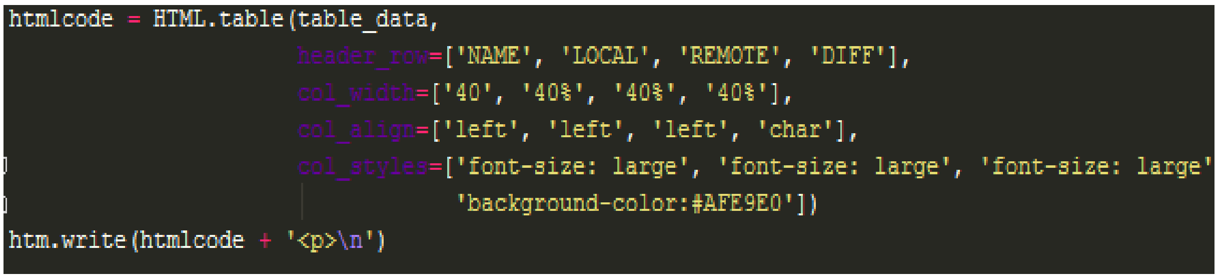
The first two lines import the HTML tools and third line opens an HTML page created in the same folder as the project and name it “results.html”. Instead of “a” could be “w” is all about the writing rights. The “w” allows the results to be written in the HTML form from the beginning, whereas “a” allows to write in the continuity of the existing text. Therefore, in the same way as the TEXT file, to write the results into an HTML form there are more than just those 3 lines. In the following figures, are captured some examples of how the project is generating the HTML report.



```
text.write("Wrong path-directory name: " + sys.argv[2])
text.flush()
t18 = ("Wrong path-directory name: " + sys.argv[2])
htm.write(HTML_tools.Title4(t18))
```

Figure 63. Check.py –HTML

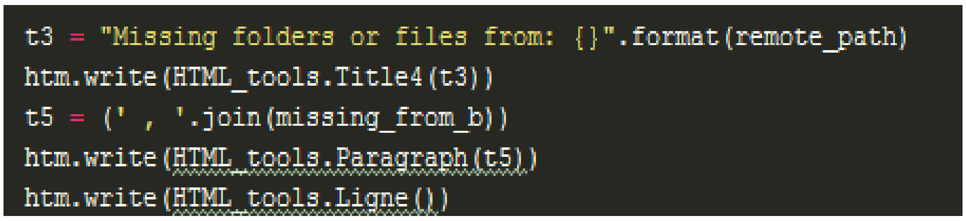
Title 4 is the method settling different titles size that HTML tools provide. It could be title 3 for bigger title size or title 5 for smaller ones.



```
htmlcode = HTML.table(table_data,
                      header_row=['NAME', 'LOCAL', 'REMOTE', 'DIFF'],
                      col_width=['40', '40%', '40%', '40%'],
                      col_align=['left', 'left', 'left', 'char'],
                      col_styles=['font-size: large', 'font-size: large', 'font-size: large'
                                  'background-color:#AFE9E0'])
htm.write(htmlcode + '<p>\n')
```

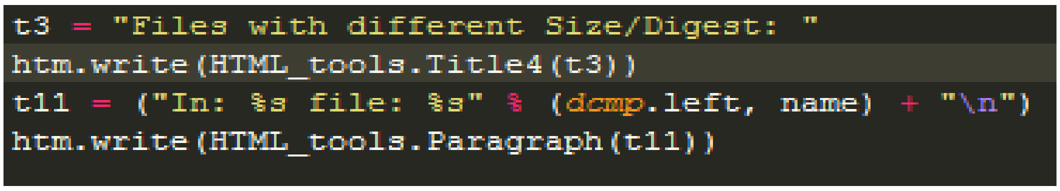
Figure 64. Statistics.py- HTML

At the figure 63 is presented an HTML table called htmlcode. It has 4 columns and the background is white. Also it uses large font size.



```
t3 = "Missing folders or files from: {}".format(remote_path)
htm.write(HTML_tools.Title4(t3))
t5 = (' , '.join(missing_from_b))
htm.write(HTML_tools.Paragraph(t5))
htm.write(HTML_tools.Ligne())
```

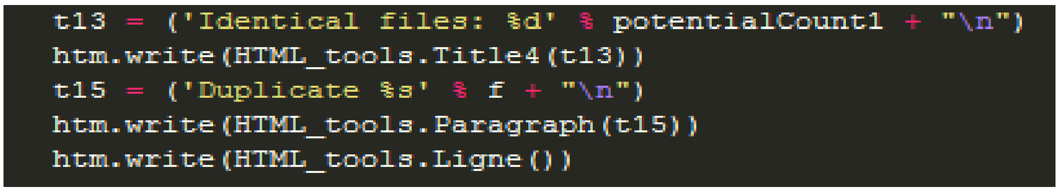
Figure 65. Bascmp.py –HTML



```
t3 = "Files with different Size/Digest: "
htm.write(HTML_tools.Title4(t3))
t11 = ("In: %s file: %s" % (dcmp.left, name) + "\n")
htm.write(HTML_tools.Paragraph(t11))
```

Figure 66. Digest.py –HTML

Here are used the methods title 4 and paragraph as it is going to display a list, meaning several written lines.



```
t13 = ('Identical files: %d' % potentialCount1 + "\n")
htm.write(HTML_tools.Title4(t13))
t15 = ('Duplicate %s' % f + "\n")
htm.write(HTML_tools.Paragraph(t15))
htm.write(HTML_tools.Ligne())
```

Figure 67. Duplicate.py –HTML

The extra code HTML_Tools_Ligne is adding a line underneath the end of the results. In this way results can be split in different groups for a better understanding.

Chapter 5. Conclusion & Future work

5.1 Conclusion

The main purpose of the current thesis is to develop a program supervising data transfers. Comparing in deep the copied data with the original, the program ensures the successfully transfer and allows the users to delete the original data. Furthermore, the user is able to detect files duplicates which unnecessarily occupies extra storage capacity. It is a tool which keeps organized the big amount of data that are daily produced.

5.2 Future work

The aforesaid program can be developed even further in the future. Firstly, it can be part of a software instead of being just a python program. Secondly, it can includes code which detects and displays where exactly the data were modified instead. Also, why not to have a side by side display of the modified data. In any ways, it can be used as it is or it can be developed even more.

6. BIBLIOGRAPHY

6.1 Links

- http://java.about.com/od/gettingstarted/a/ideversuseditor.htm
- http://www.sixfeetup.com/blog/why-we-choose-python
- http://www.udemy.com/blog/modern-language-wars/
- http://itpings.com/category/development/
- http://en.wikipedia.org/wiki/SHA-1#The_SHA-1_hash_function
- http://www.xorbin.com/tools/sha256-hash-calculator
- http://www.tutorialspoint.com/python/os_lstat.htm
- http://pythoncard.sourceforge.net/what_is_python.html
- http://en.wikipedia.org/wiki/MD5
- https://docs.python.org/3/
- https://www.jetbrains.com/pycharm/download/#section=windows
- https://www.python.org/doc/essays/comparisons/

6.2 Books

- Learning to Program Using Python
- Think Python: An Introduction to Software Design
- An Introduction to Python
- A Byte of Python
- Learn Python the hard way

Chapter 7. Appendixes

7.1 Check.py

```
import os.path
import subprocess
import sys
import HTML_tools
from list import *
import time
text = open("./results.txt", "w")
htm = open("./results.html", 'w')
def main():
    pass
start = time.time()
if len(sys.argv) != 3 and len(sys.argv) != 1:
    print "Usage: py name.py  /Local_path /Remote_path"
else:
    if len(sys.argv) == 3:
        local_path = sys.argv[1]
        remote_path = sys.argv[2]
        if not os.path.isdir(sys.argv[1]):
            print "Wrong path-directory name: " + sys.argv[1]
            text.write("Wrong path-directory name: " + sys.argv[1])
            text.flush()
            t17 = ("Wrong path-directory name: " + sys.argv[1])
            htm.write(HTML_tools.Title4(t17))
        if not os.path.exists(sys.argv[2]):
            print "Wrong path-directory name: " + sys.argv[2]
            text.write("Wrong path-directory name: " + sys.argv[2])
            text.flush()
            t18 = ("Wrong path-directory name: " + sys.argv[2])
            htm.write(HTML_tools.Title4(t18))
        else:
            subprocess.call(" py statistic.py " + local_path + " " + remote_path, shell=True)
            subprocess.call(" py duplicate.py " + local_path + " " + remote_path, shell=True)
            subprocess.call(" py comparison.py " + local_path + " " + remote_path, shell=True)
    if len(sys.argv) == 1:
        size = len(local_path1)
        for i in range(size):
            local_path = local_path1[i]
            remote_path = remote_path1[i]

            if not os.path.isdir(local_path):
                print "Wrong path-directory name: " + local_path
                text.write("Wrong path-directory name: " + local_path)
                text.flush()
                t17 = ("Wrong path-directory name: " + local_path)
                htm.write(HTML_tools.Title4(t17))

            if not os.path.exists(remote_path):
                print "Wrong path-directory name: " + remote_path
                text.write("Wrong path-directory name: " + remote_path)
                text.flush()
                t18 = ("Wrong path-directory name: " + remote_path)
                htm.write(HTML_tools.Title4(t18))
            else:
                subprocess.call(" py statistic.py " + local_path + " " + remote_path, shell=True)
                subprocess.call(" py duplicate.py " + local_path + " " + remote_path, shell=True)
                subprocess.call(" py comparison.py " + local_path + " " + remote_path, shell=True)

end = time.time()
print(end - start)

text.close()

if __name__ == '__main__':
    main()
```

7.2 Statistic.py

```
import os
import os.path
import sys
import HTML
import HTML_tools

text = open("./results.txt", "a")
htm = open("./results.html", 'a')

local_path = sys.argv[1]
remote_path = sys.argv[2]

def getFolderSize(path):
    total_size = os.path.getsize(path)
    for item in os.listdir(path):
        itempath = os.path.join(path, item)
        if os.path.isfile(itempath):
            total_size += os.path.getsize(itempath)
        elif os.path.isdir(itempath):
            total_size += getFolderSize(itempath)
    return total_size
totalc = 0
totac = 0
totc = 0
for root, dirs, files in os.walk(remote_path):
    totalc += len(files)
    totac += len(files)
    totalc += len(dirs)
    totc += len(dirs)

totalo = 0
totao = 0
toto = 0
```

```
for [root, dirs, files] in os.walk(local_path):
    totalo += len(files)
    totao += len(files)
    totalo += len(dirs)
    toto += len(dirs)

t_size_local_path = float(getFolderSize(local_path))
x = "%0.1f " % (t_size_local_path / (1024 * 1024 * 1024.0))

t_size_remote_path = float(getFolderSize(remote_path))
y = "%0.1f " % (t_size_remote_path / (1024 * 1024 * 1024.0))

a = abs(totalo - totalc)
b = abs(toto - totc)
c = abs(totao - totac)
d = abs(t_size_local_path - t_size_remote_path)
e = float(float(t_size_local_path / (1024 * 1024 * 1024.0)) - float(t_size_remote_path / (1024 * 1024 * 1024.0)))
f = "%0.4f" % abs(e)

print "Comparing: " + local_path + " with:" + remote_path + "\n"
text.write("Comparing:  " + local_path + " with: " + remote_path + "\n")
text.flush()

t1 = "Comparing:  " + local_path + "  with: " + remote_path + "\n"
htm.write(HTML_tools.Title3(t1))
htm.write(HTML_tools.Ligne())

d1 = [
    ['PATH', [('LOCAL:', local_path), ('REMOTE:', remote_path), ('DIFF:', "")]],
    ['ITEMS', [('LOCAL:', totalo), ('REMOTE:', totalc), ('DIFF:', a)]],
    ['DIRECTORIES', [('LOCAL:', toto), ('REMOTE:', totc), ('DIFF:', b)]],
    ['FILES', [('LOCAL:', totao), ('REMOTE:', totac), ('DIFF:', c)]],
    ['T.SIZE/Byte', [('LOCAL:', t_size_local_path), ('REMOTE:', t_size_remote_path), ('DIFF:', d)]],
    ['GB', [('LOCAL:', x), ('REMOTE:', y), ('DIFF:', f)]],
]
```

```
COLUMN_WIDTH = 10
STRING_WHEN_MISSING = '""'
PADDING_STRING = '.'
labelSize = 21
labelColumn = ''
labels = ['NAME', 'LOCAL', 'REMOTE', 'DIFF']
labelColumn += labels[0] + (labelSize - len(labels[0])) * PADDING_STRING + 2 * ' '
labelColumn += labels[1] + (labelSize - len(labels[1])) * PADDING_STRING + 2 * ' '
labelColumn += labels[2] + (labelSize - len(labels[2])) * PADDING_STRING + 2 * ' '
labelColumn += labels[3] + (labelSize - len(labels[3]) - 11) * PADDING_STRING + '*'
labelColumn += '\n'
for (row) in d1:
    rowId = row[0]
    valueLocal = row[1][0][1]
    valueRemote = row[1][1][1]
    valueDIFF = row[1][2][1]

    labelColumn += rowId + (labelSize - len(rowId)) * PADDING_STRING + 2 * ' '
    labelColumn += str(valueLocal) + (labelSize - len(str(valueLocal))) * PADDING_STRING + 2 * ' '
    labelColumn += str(valueRemote) + (labelSize - len(str(valueRemote))) * PADDING_STRING + 2 * ' '
    labelColumn += str(valueDIFF) + (labelSize - len(str(valueDIFF)) - 11) * PADDING_STRING + '*'
    labelColumn += '\n'

print'                              Statistical Table'
text.write("\n" + '                              Statistical Table' + "\n")
text.flush()

t2 = 'Statistical Table'
htm.write(HTML_tools.center(t2))

print labelColumn
text.write(labelColumn + "\n")
text.flush()
table_data = [
    ['PATH', local_path, remote_path, ""],
    ['ITEMS', totalo, totalc, a],
    ['DIRECTORIES', toto, totc, b],
    ['FILES', totao, totac, c],
    ['T.SIZE/Byte', t_size_local_path, t_size_remote_path, d],
    ['GB', x, y, f]
]
htmlcode = HTML.table(table_data,
                      header_row=['NAME', 'LOCAL', 'REMOTE', 'DIFF'],
                      col_width=['40', '40%', '40%', '40%'],
                      col_align=['left', 'left', 'left', 'char'],
                      col_styles=['font-size: large', 'font-size: large', 'font-size: large',
                                  'background-color:#AFE9E0'])
htm.write(htmlcode + '<p>\n')
```

7.3 Basicmp.py

```
import HTML_tools
import sys
import os

text = open("./results.txt", "a")
htm = open("./results.html", 'a')

local_path = sys.argv[1]
remote_path = sys.argv[2]

folder_a = set(os.listdir(local_path))
folder_b = set(os.listdir(remote_path))

missing_from_a = folder_a - folder_b
missing_from_b = folder_b - folder_a

if not missing_from_a:
    print("NO missing folders or files from: {}".format(local_path)) + "\n"
    text.write("NO missing folders or files from: " + local_path + "\n")

    t3 = "No missing folders or files from: {}".format(local_path)
    htm.write(HTML_tools.Title4(t3))

else:
    print "Missing folders or files from: {}".format(local_path)
    print ', '.join(missing_from_a) + "\n"

    text.write('Missing folders or files from:' + local_path + "\n")
    text.write(",".join(missing_from_a))
    text.write("\n")

    t3 = "Missing folders or files from: {}".format(local_path)
    htm.write(HTML_tools.Title4(t3))
    t5 = (' , '.join(missing_from_a))
    htm.write(HTML_tools.Paragraph(t5))
    htm.write(HTML_tools.Ligne())

text.write("\n")
if not missing_from_b:
    print("No missing folders or files from: {}".format(remote_path)) + "\n"
    text.write("No missing folders or files from: " + remote_path + "\n")
    t3 = "No missing folders or files from: {}".format(remote_path)
    htm.write(HTML_tools.Title4(t3))

else:
    print "Missing folders or files from: {}".format(remote_path)
    print ', '.join(missing_from_b) + "\n"

    text.write("Missing folders or files from: " + remote_path + "\n")
    text.write(",".join(missing_from_b))
    text.write("\n")

    t3 = "Missing folders or files from: {}".format(remote_path)
    htm.write(HTML_tools.Title4(t3))
    t5 = (' , '.join(missing_from_b))
    htm.write(HTML_tools.Paragraph(t5))
    htm.write(HTML_tools.Ligne())

text.write("\n")
```

7.4 Digest.py

```
import sys
from filecmp import dircmp
import HTML_tools

text = open("./results.txt", "a")
htm = open("./results.html", 'a')

local_path = sys.argv[1]
remote_path = sys.argv[2]

print "Files with different Size/Digest: "
text.write("Files with different Size/Digest: " + "\n")
t3 = "Files with different Size/Digest: "
htm.write(HTML_tools.Title4(t3))

def print_diff_files(dcmp):
    for name in dcmp.diff_files:
        print "file: %s" % (name)

        text.write("In: %s file: %s" % (dcmp.left, name) + "\n")
        text.flush()

        t3 = "Files with different Size/Digest: "
        htm.write(HTML_tools.Title4(t3))
        t11 = ("In: %s file: %s" % (dcmp.left, name) + "\n")
        htm.write(HTML_tools.Paragraph(t11))
    for sub_dcmp in dcmp.subdirs.values():
        print_diff_files(sub_dcmp)



dcmp = dircmp(local_path, remote_path)
print_diff_files(dcmp)
text.close()
```

7.5 Duplicate.py

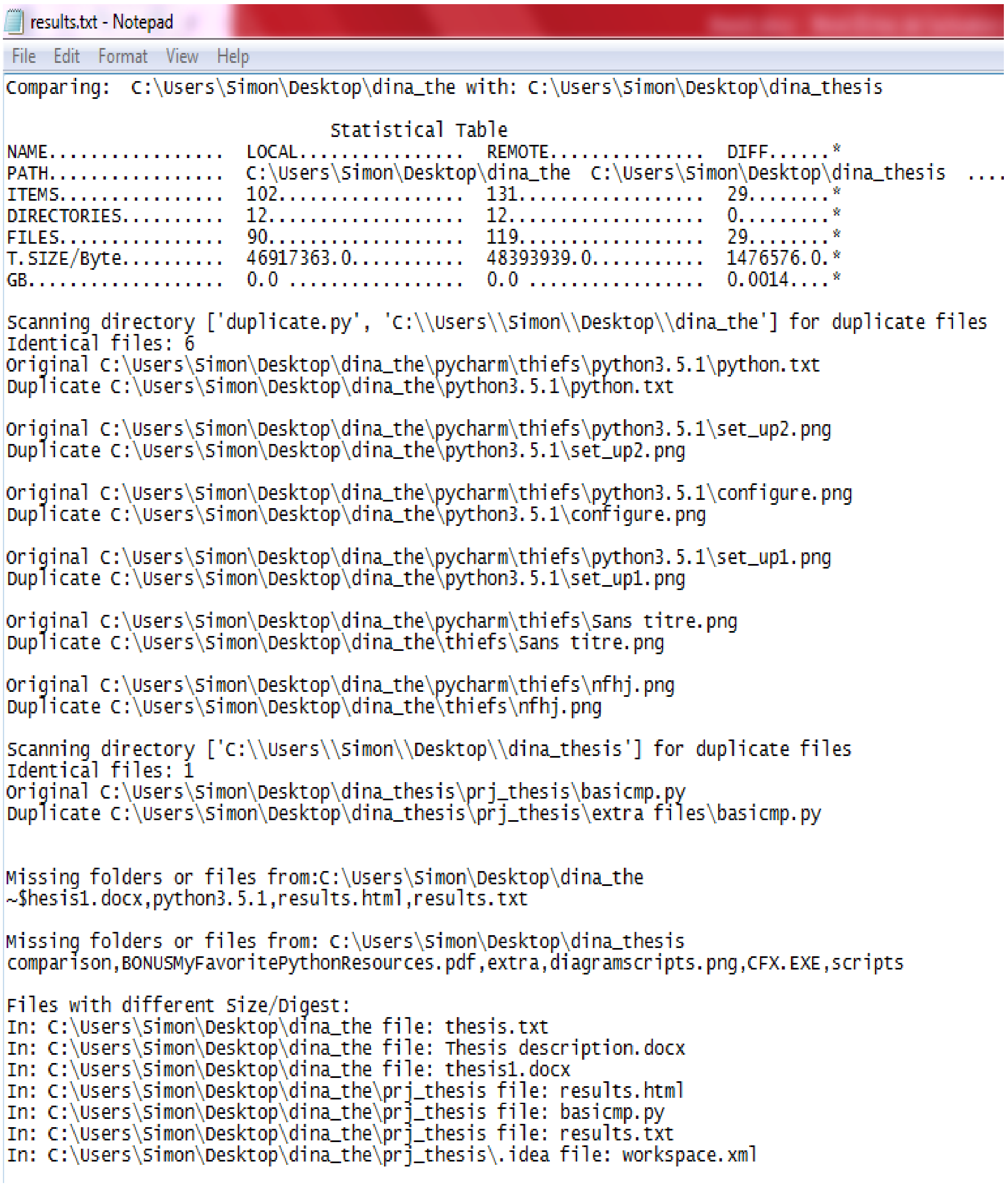
```
import md5
import os
import os.path
import stat
import sys
import HTML_tools

text = open("./results.txt", "a")
htm = open("./results.html", 'a')

local_path = sys.argv[1]
remote_path = sys.argv[2]



def duplicate(path):
    filesBySize = {}

    def walker(arg, dirname, fnames):
        d = os.getcwd()
        os.chdir(dirname)
        try:
            fnames.remove('Thumbs')
        except ValueError:
            pass
        for f in fnames:
            if not os.path.isfile(f):
```

```
            continue
        size = os.stat(f)[stat.ST_SIZE]
        if size < 100:
            continue
        if filesBySize.has_key(size):
            a = filesBySize[size]
        else:
            a = []
            filesBySize[size] = a
        a.append(os.path.join(dirname, f))
    os.chdir(d)

for x in path:
    os.path.walk(x, walker, filesBySize)

potentialDupes = []
potentialCount = 0
potentialCount1 = 0
trueType = type(True)
sizes = filesBySize.keys()
sizes.sort()
for k in sizes:
    inFiles = filesBySize[k]
    outFiles = []
    hashes = {}
    #     if len(inFiles) is 1: continue
    #     print 'Testing %d files of size %d...' % (len(inFiles), k)
    for fileName in inFiles:
        if not os.path.isfile(fileName):
            continue
        aFile = file(fileName, 'r')
        hasher = md5.new(aFile.read(1024))
        hashValue = hasher.digest()
        if hashes.has_key(hashValue):
            x = hashes[hashValue]
            if type(x) is not trueType:
                outFiles.append(hashes[hashValue])
                hashes[hashValue] = True
            outFiles.append(fileName)
        else:
            hashes[hashValue] = fileName
        aFile.close()
    if len(outFiles):
        potentialDupes.append(outFiles)
        potentialCount += len(outFiles)
        potentialCount1 = potentialCount / 2

print 'Scanning directory {0:s}  for duplicate files'.format(path)
text.write('Scanning directory {0:s} for duplicate files'.format(path) + '\n')
t12 = ('Scanning directory for duplicates files in: {}'.format(path))
htm.write(HTML_tools.Title4(t12))


print 'Identical files: %d' % potentialCount1
text.write('Identical files: %d' % potentialCount1)
text.write("\n")
t13 = ('Identical files: %d' % potentialCount1 + "\n")
htm.write(HTML_tools.Title4(t13))

dupes = []
for aSet in potentialDupes:
    outFiles = []
    hashes = {}
    for fileName in aSet:
        aFile = file(fileName, 'r')
        hasher = md5.new()
        while True:
            r = aFile.read(4096)
            if not len(r):
                break
            hasher.update(r)
            aFile.close()
            hashValue = hasher.digest()
            if hashes.has_key(hashValue):
                if not len(outFiles):
                    outFiles.append(hashes[hashValue])
                outFiles.append(fileName)
            else:
                hashes[hashValue] = fileName
        if len(outFiles):
            dupes.append(outFiles)

    i = 0

    for d in dupes:
        print 'Original %s' % d[0]
        text.write('Original %s' % d[0] + "\n")
        text.flush()
        t14 = ('Original %s' % d[0] + "\n")
        htm.write(HTML_tools.Paragraph(t14))

        for f in d[1:]:
            i = i + 1
            print 'Duplicate %s' % f + "\n"
            text.write('Duplicate %s' % f + "\n")
            text.write("\n")
            text.flush()
            t15 = ('Duplicate %s' % f + "\n")
            htm.write(HTML_tools.Paragraph(t15))
            htm.write(HTML_tools.Ligne())

print "\n"
if sys.argv[1:]:
    duplicate(sys.argv[:2])
    duplicate(sys.argv[2:])
text.write("\n")
```

7.6 List.py

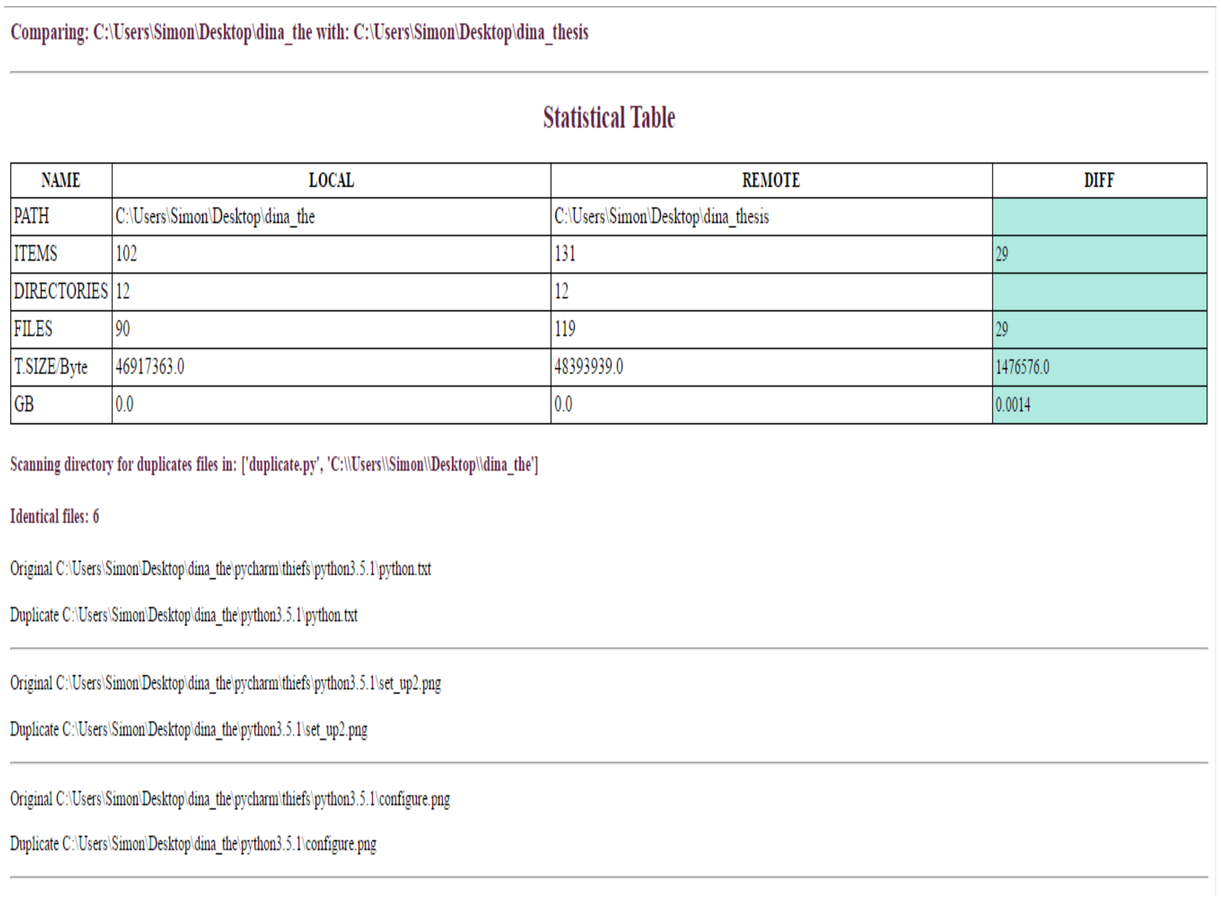
```
local_path1 = ["C:\Users\Simon\Desktop\dina_the"
              ]
remote_path1 = ["C:\Users\Simon\Desktop\dina_thesis"
               ]
```

7.8 Results TXT



```
Comparing:  C:\Users\Simon\Desktop\dina_the with: C:\Users\Simon\Desktop\dina_thesis

                              Statistical Table
NAME................  LOCAL...............  REMOTE..............  DIFF......*
PATH................  C:\Users\Simon\Desktop\dina_the  C:\Users\Simon\Desktop\dina_thesis  ....
ITEMS...............  102.................  131.................  29........*
DIRECTORIES.........  12..................  12..................  0.........*
FILES...............  90..................  119.................  29........*
T.SIZE/Byte.........  46917363.0..........  48393939.0..........  1476576.0.*
GB..................  0.0 ................  0.0 ................  0.0014....*

Scanning directory ['duplicate.py', 'C:\\Users\\Simon\\Desktop\\dina_the'] for duplicate files
Identical files: 6
Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\python.txt
Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\python.txt

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\set_up2.png
Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\set_up2.png

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\configure.png
Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\configure.png

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\set_up1.png
Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\set_up1.png

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\Sans titre.png
Duplicate C:\Users\Simon\Desktop\dina_the\thiefs\Sans titre.png

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\nfhj.png
Duplicate C:\Users\Simon\Desktop\dina_the\thiefs\nfhj.png

Scanning directory ['C:\\Users\\Simon\\Desktop\\dina_thesis'] for duplicate files
Identical files: 1
Original C:\Users\Simon\Desktop\dina_thesis\prj_thesis\basicmp.py
Duplicate C:\Users\Simon\Desktop\dina_thesis\prj_thesis\extra files\basicmp.py


Missing folders or files from:C:\Users\Simon\Desktop\dina_the
~$hesis1.docx,python3.5.1,results.html,results.txt

Missing folders or files from: C:\Users\Simon\Desktop\dina_thesis
comparison,BONUSMyFavoritePythonResources.pdf,extra,diagramscripts.png,CFX.EXE,scripts

Files with different Size/Digest:
In: C:\Users\Simon\Desktop\dina_the file: thesis.txt
In: C:\Users\Simon\Desktop\dina_the file: Thesis description.docx
In: C:\Users\Simon\Desktop\dina_the file: thesis1.docx
In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.html
In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: basicmp.py
In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.txt
In: C:\Users\Simon\Desktop\dina_the\prj_thesis\.idea file: workspace.xml
```

7.8 Results HTML



Comparing: C:\Users\Simon\Desktop\dina_the with: C:\Users\Simon\Desktop\dina_thesis

Statistical Table

NAME	LOCAL	REMOTE	DIFF
PATH	C:\Users\Simon\Desktop\dina_the	C:\Users\Simon\Desktop\dina_thesis	
ITEMS	102	131	29
DIRECTORIES	12	12	
FILES	90	119	29
T.SIZE/Byte	46917363.0	48393939.0	1476576.0
GB	0.0	0.0	0.0014

Scanning directory for duplicates files in: ['duplicate.py', 'C:\\Users\\Simon\\Desktop\\dina_the']

Identical files: 6

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\python.txt

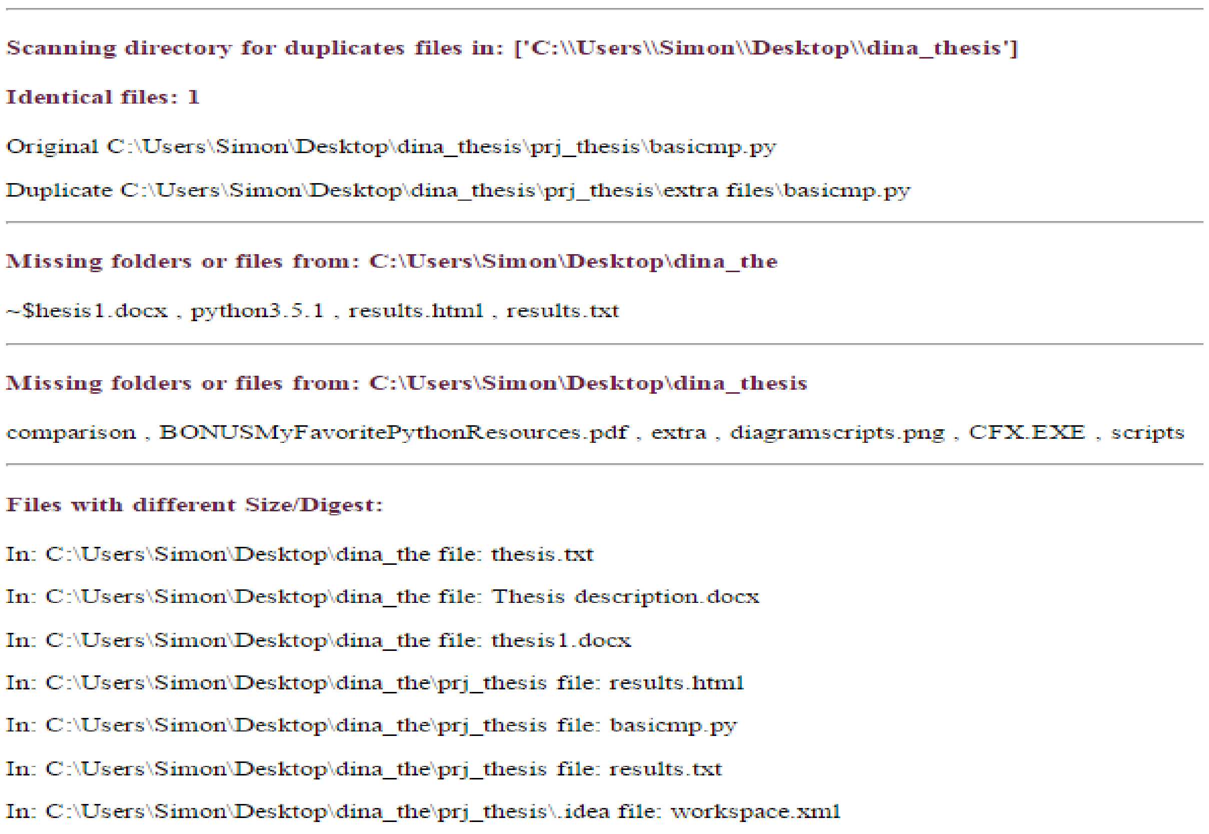
Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\python.txt

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\set_up2.png

Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\set_up2.png

Original C:\Users\Simon\Desktop\dina_the\pycharm\thiefs\python3.5.1\configure.png

Duplicate C:\Users\Simon\Desktop\dina_the\python3.5.1\configure.png



Scanning directory for duplicates files in: ['C:\\Users\\Simon\\Desktop\\dina_thesis']

Identical files: 1

Original C:\Users\Simon\Desktop\dina_thesis\prj_thesis\basicmp.py

Duplicate C:\Users\Simon\Desktop\dina_thesis\prj_thesis\extra files\basicmp.py

Missing folders or files from: C:\Users\Simon\Desktop\dina_the

~$hesis1.docx , python3.5.1 , results.html , results.txt

Missing folders or files from: C:\Users\Simon\Desktop\dina_thesis

comparison , BONUSMyFavoritePythonResources.pdf , extra , diagramscripts.png , CFX.EXE , scripts

Files with different Size/Digest:

In: C:\Users\Simon\Desktop\dina_the file: thesis.txt

In: C:\Users\Simon\Desktop\dina_the file: Thesis description.docx

In: C:\Users\Simon\Desktop\dina_the file: thesis1.docx

In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.html

In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: basicmp.py

In: C:\Users\Simon\Desktop\dina_the\prj_thesis file: results.txt

In: C:\Users\Simon\Desktop\dina_the\prj_thesis\.idea file: workspace.xml