**Aim:**

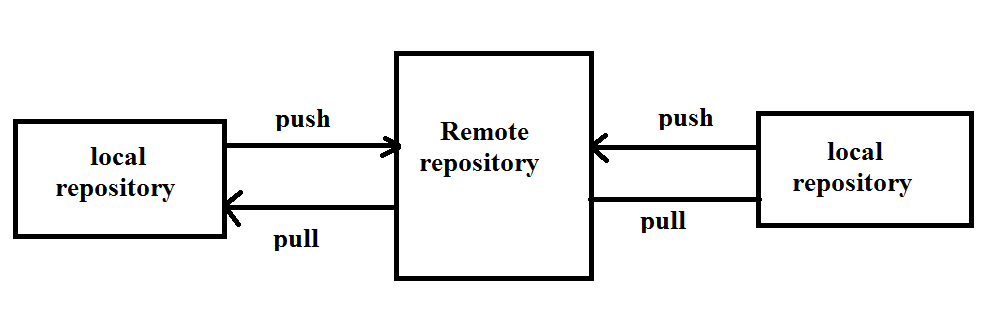
* Using Github for project management.
* Understanding the Concept of OSI model and TCP/IP protocol suite.
* Understanding the Concept of NumPy and SciPy packages in Python.

**Theory:**

1. **Project management using Github**

Git is a Distributed Version Control System(DVCS). "Distributed" means that all developers within a team will have a complete version of the project. A version control system is simply software that lets us effectively manage application versions. The purpose of git is to manage a project, or a set of files. GitHub is a code hosting platform for version control and collaboration. It lets us work together on projects from anywhere. It is a version control tool that will allow you to perform all kinds of operations to fetch dat from the central server or push data to it.

It merges the code from different computers and different team members.





1. **OSI model and TCP/IP Protocol**

**2.1. OSI model**

**3. NumPy and SciPy packages in Python.**

**3.1 Python**

The idea of Python originated in 1989 when its creator Guido van Rossum was confronted by the shortcomings of ABC language (namely extensibility). Rossum started work on developing a new language that integrated all good features of ABC language and new desired features, such as extensibility and exception handling. Python 1.0 was released in 1994; it borrowed the module system from Modula-3, had the capability to interact with Amoeba operating system, and included functional programming tools.

In 2000, Python’s core development team moved to Beopen.com, and in October 2000, **Python 2.0** was released with many improvisations including a garbage collector and support for Unicode.

December 2008 saw the release of **Python 3.0,** giving up backward compatibility and possessing a new design to avoid duplicative constructs and modules. It is still a multi-paradigm language offering developers the options of object-orientation, structured programming and functional programming.

Python today has multiple implementations including Jython,scripted in Java language for Java Virtual Machine*;*IronPythonwritten in C# for the Common Language Infrastructure, and PyPyversionwritten in RPython and translated into C. To be noted, Cpythonwhich is written in C and developed by Python Software Foundation is the default and most popular implementation of Python. While these implementations work in the native language they are written in, they are also capable of interacting with other languages through use of modules. Most of these modules work on community development model and are open-source and free.

Released in February 2015, Python 3.4.3 offers drastic improvement in Unicode support, among other new features. Python 3.5 is currently in development, with scheduled release in September 2015.

Python is an object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding; make 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 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.

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, and 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. Python is much better 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.

**3.1.1 Advantages of using python**

**Easy Syntax**

Python's syntax is easy to learn, so both non-programmers and programmers can start programming right away.

**Readability**

Python's syntax is very clear, so it is easy to understand program code. (Python is often referred to as "executable pseudo-code" because its syntax mostly follows the conventions used by programmers to outline their ideas without the formal verbosity of code in most programming languages; in other words syntax of Python is almost identical to the simplified "pseudo-code" used by many programmers to prototype and describe their solution to other programmers. Thus Python can be used to prototype and test code which is later to be implemented in other programming languages)..

**High-Level Language**

Python looks more like a readable, human language than like a low-level language. This gives you the ability to program at a faster rate than a low-level language will allow you.

**Object oriented programming**

Object-oriented programming allows you to create data structures that can be re-used, which reduces the amount of repetitive work that you'll need to do. Programming languages usually define objects with namespaces, like class or def, and objects can edit themselves by using keyword, like this or self. Most modern programming languages are object-oriented (such as Java, C++, and C#) or have support for OOP features (such as Perl version 5 and later). Additionally object-oriented techniques can be used in the design of almost any non-trivial software and implemented in almost any programming or scripting language. (For example a number of Linux kernel features are "objects" which implement their own [encapsulation](https://en.wikipedia.org/wiki/Encapsulation_(object-oriented_programming)" \o "wikipedia:Encapsulation (object-oriented programming)) of behavior and data structive via pointers, specifically pointers to functions, in the C programming language).[*[citation needed](https://en.wikiversity.org/wiki/Wikiversity:Citation_needed" \o "Wikiversity:Citation needed)*] Python's support for object-oriented programming is one of its greatest benefits to new programmers because they will be encountering the same concepts and terminology in their work environment. If you ever decide to switch languages, or use any other for that fact, you'll have a significant chance that you'll be working with object-oriented programming.[[1]](https://en.wikiversity.org/wiki/Python_Concepts/Why_learn_Python" \l "cite_note-1)

**It's Free**

Python is both free and open-source. The Python Software Foundation distributes pre-made binaries that are freely available for use on all major operating systems called CPython. You can get CPython's source-code, too. Plus, you can modify the source code and distribute as allowed by CPython's license. [[2]](https://en.wikiversity.org/wiki/Python_Concepts/Why_learn_Python" \l "cite_note-2) (Luckily, CPython has a [permissive free software license](https://en.wikipedia.org/wiki/Permissive_free_software_licence" \o "wikipedia:Permissive free software licence) attitude.)

**Cross-platform**

Python runs on all major operating systems like Microsoft Windows, Linux, and Mac OS X.

**Widely Supported**

Python has an active support community with many web sites, mailing lists, and USENET "netnews" groups that attract a large number of knowledgeable and helpful contributes.

**It's Safe**

Python doesn't have pointers like other C-based languages, making it much more reliable. Along with that, errors never pass silently unless they're explicitly silenced. This allows you to see and read why the program crashed and where to correct your error.

**Batteries Included**

Python is famous for being the "batteries are included" language. There are over 300 standard library modules which contain modules and classes for a wide variety of programming tasks. For example the standard library contains modules for safely creating temporary files (named or anonymous), mapping files into memory (including use of shared and anonymous memory mappings), spawning and controlling sub-processes, compressing and decompressing files (compatible with gzip or PK-zip) and archives files (such as Unix/Linux "tar"), accessing indexed "DBM" (database) files, interfacing to various graphical user interfaces (such as the TK toolkit and the popular WxWindows multi-platform windowing system), parsing and maintaining CSV (comma-separated values) and ".cfg" or ".ini" configuration files (similar in syntax to the venerable WIN.INI files from MS-DOS and MS-Windows), for sending e-mail, fetching and parsing web pages, etc. It's possible, for example, to create a custom web server in Python using less than a dozen lines of code, and one of the standard libraries, of course.

**Extensible**

In addition to the standard libraries there are extensive collections of freely available add-on modules, libraries, frameworks, and tool-kits. These generally conform to similar standards and conventions; for example almost all of the database adapters (to talk to almost any client-server RDBMS engine such as MySQL, Postgres, Oracle, etc) conform to the Python DBAPI and thus can mostly be accessed using the same code. So it's usually easy to modify a Python program to support any database engine.

3.1.2 Applications

* GUI based desktop applications
  + Image processing and graphic design applications
  + Scientific and computational applications
  + Games
* Web frameworks and web applications
* Enterprise and business applications
* Operating systems
* Language development
* Prototyping

**3.2. NumPy**

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

* NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
* The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
* NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
* A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today’s scientific/mathematical Python-based software, just knowing how to use Python’s built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

The points about sequence size and speed are particularly important in scientific computing. As a simple example, consider the case of each element in a 1-D sequence with the corresponding element in another sequence of the same length. If the data are stored in two Python lists, a and b, we could iterate over each element:

c = [] for i in range(len(a)): c.append(a[i]\*b[i])

This produces the correct answer, but if a and b each contain millions of numbers, we will pay the price for the inefficiencies of looping in Python. We could accomplish the same task much more quickly in C by writing (for clarity we neglect variable declarations and initializations, memory allocation, etc.)

for (i = 0; i < rows; i++): { c[i] = a[i]\*b[i]; }

This saves all the overhead involved in interpreting the Python code and manipulating Python objects, but at the expense of the benefits gained from coding in Python. Furthermore, the coding work required increases with the dimensionality of our data. In the case of a 2-D array, for example, the C code (abridged as before) expands to

for (i = 0; i < rows; i++): { for (j = 0; j < columns; j++): { c[i][j] = a[i][j]\*b[i][j]; } }

NumPy gives us the best of both worlds: element-by-element operations are the “default mode” when an ndarray is involved, but the element-by-element operation is speedily executed by pre-compiled C code. In NumPy

c = a \* b

does what the earlier examples do, at near-C speeds, but with the code simplicity we expect from something based on Python. Two of NumPy’s features which are the basis of much of its power: vectorization and broadcasting. Vectorization describes the absence of any explicit looping, indexing, etc., in the code - these things are taking place, of course, just “behind the scenes” in optimized, pre-compiled C code. Vectorized code has many advantages, among which are:

* vectorized code is more concise and easier to read
* fewer lines of code generally means fewer bugs
* the code more closely resembles standard mathematical notation (making it easier, typically, to correctly code mathematical constructs)
* vectorization results in more “Pythonic” code. Without vectorization, our code would be littered with inefficient and difficult to read for loops.

Broadcasting is the term used to describe the implicit element-by-element behaviour of operations; generally speaking, in NumPy all operations, not just arithmetic operations, but logical, bit-wise, functional, etc., behave in this implicit element-by-element fashion, i.e., they broadcast. Moreover, in the example above, a and b could be multidimensional arrays of the same shape, or a scalar and an array, or even two arrays of with different shapes, provided that the smaller array is “expandable” to the shape of the larger in such a way that the resulting broadcast is unambiguous.

**3.3 FPGA**

Field programmable gate arrays (FPGAs) are digital integrated circuits (ICs) that  
contain configurable (programmable) blocks of logic along with configurable  
interconnects between these blocks. Design engineers can configure, or program, such devices to perform a number of tasks**. [FPGA WORLD CLASS DESIGN BOOK]**

They can be configured by a customer or a designer after manufacturing – hence "[field-programmable](https://en.wikipedia.org/wiki/Field-programmable" \o "Field-programmable)".