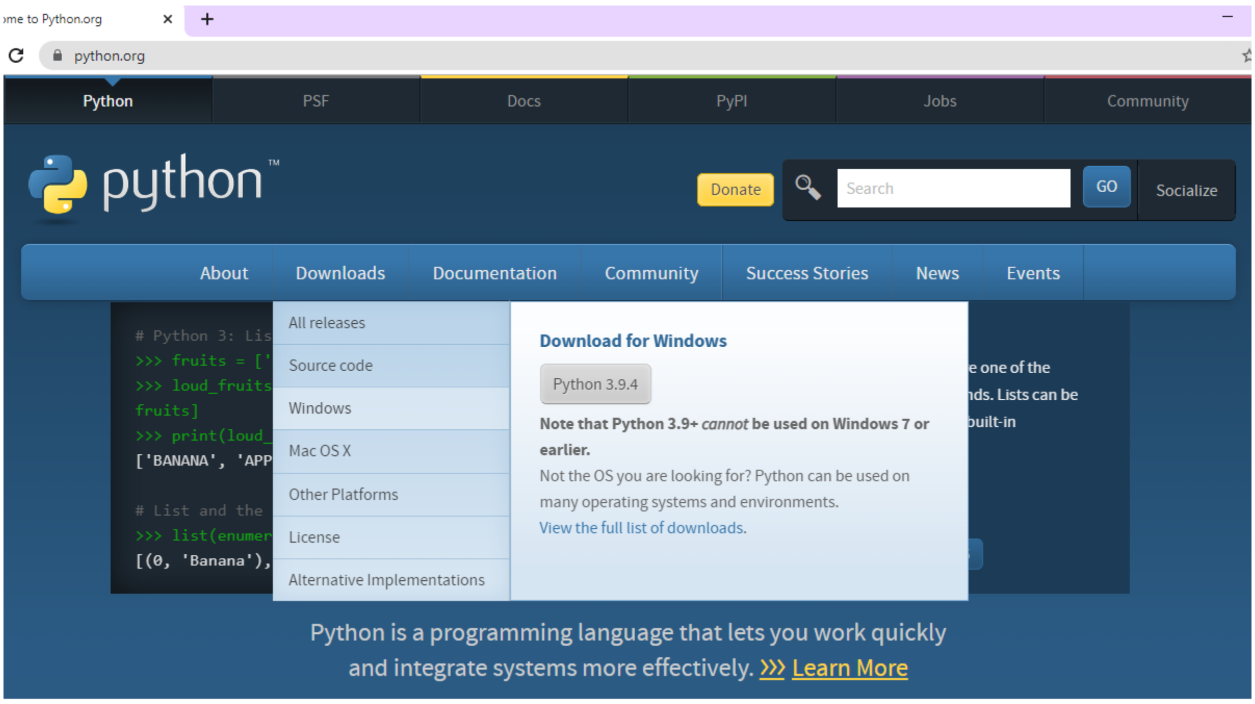
Installing Python for Windows

Installation of Python for Windows

To install Python software into our laptop or desktop, we need to visit their official website by typing ‘python.org’ in the web browser. The following screen as shown in Figure 1.4 will appear. Select ‘Downloads’ and then click on ‘Windows’ to see various versions of Python available.

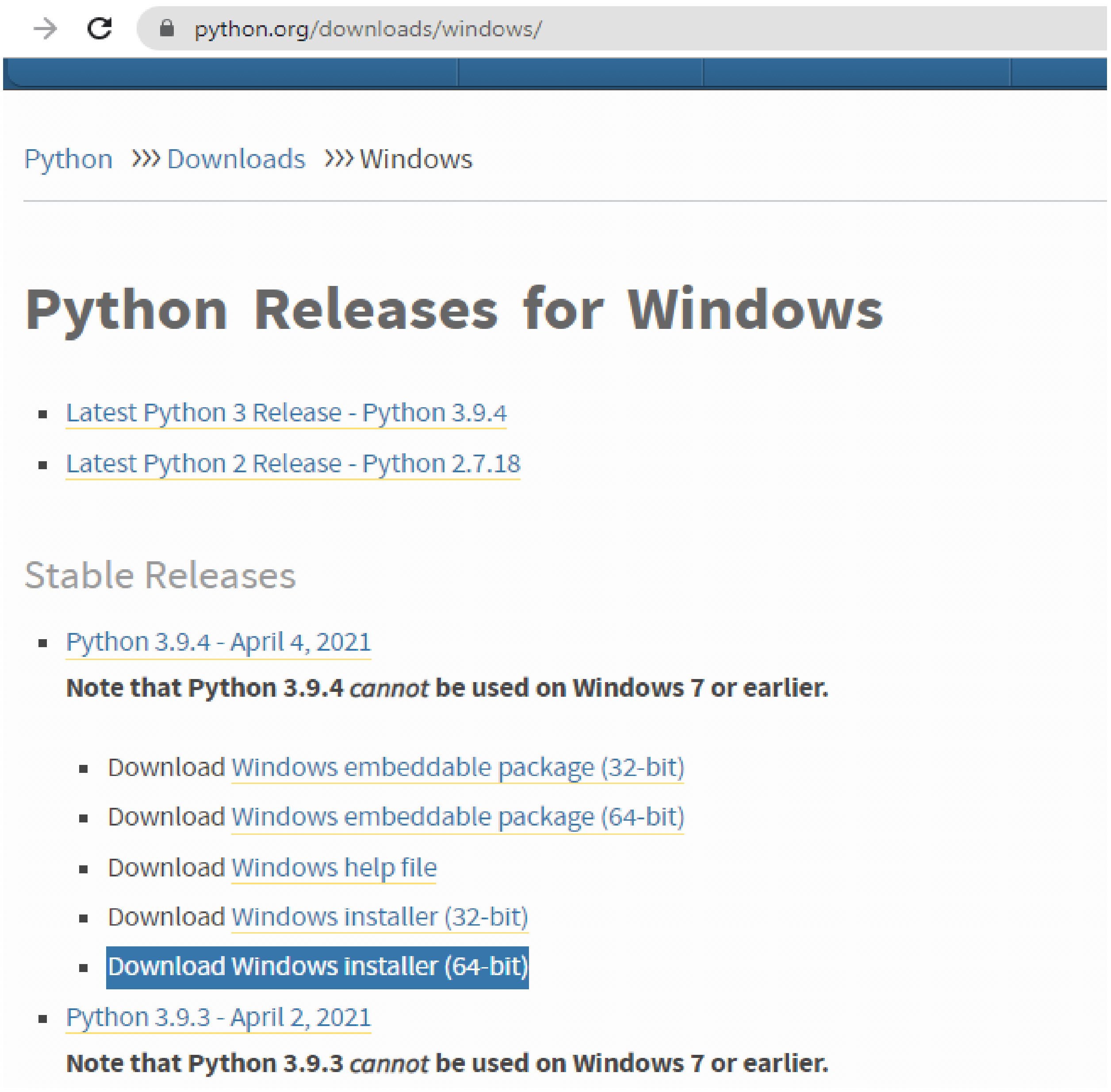


Note that Python 3.9+ *cannot* be used on Windows 7 or earlier.
Not the OS you are looking for? Python can be used on many operating systems and environments.
View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Figure 1.4: Python software download

The latest version of Python (during April of 2021) is Python 3.9.4. This version has again got two variations, a 32-bit version and a 64-bit version. Depending upon our operating system, we can choose a version. Nowadays most people use 64-bit operating system and hence we can use the 64-bit version of Python. For Machine learning and Data Science related tasks, it is better to go for 64-bit version. Hence click on ‘Download Windows installer (64-bit)’ . See Figure 1.5.


python.org/downloads/windows/

Python >>> Downloads >>> Windows

Python Releases for Windows

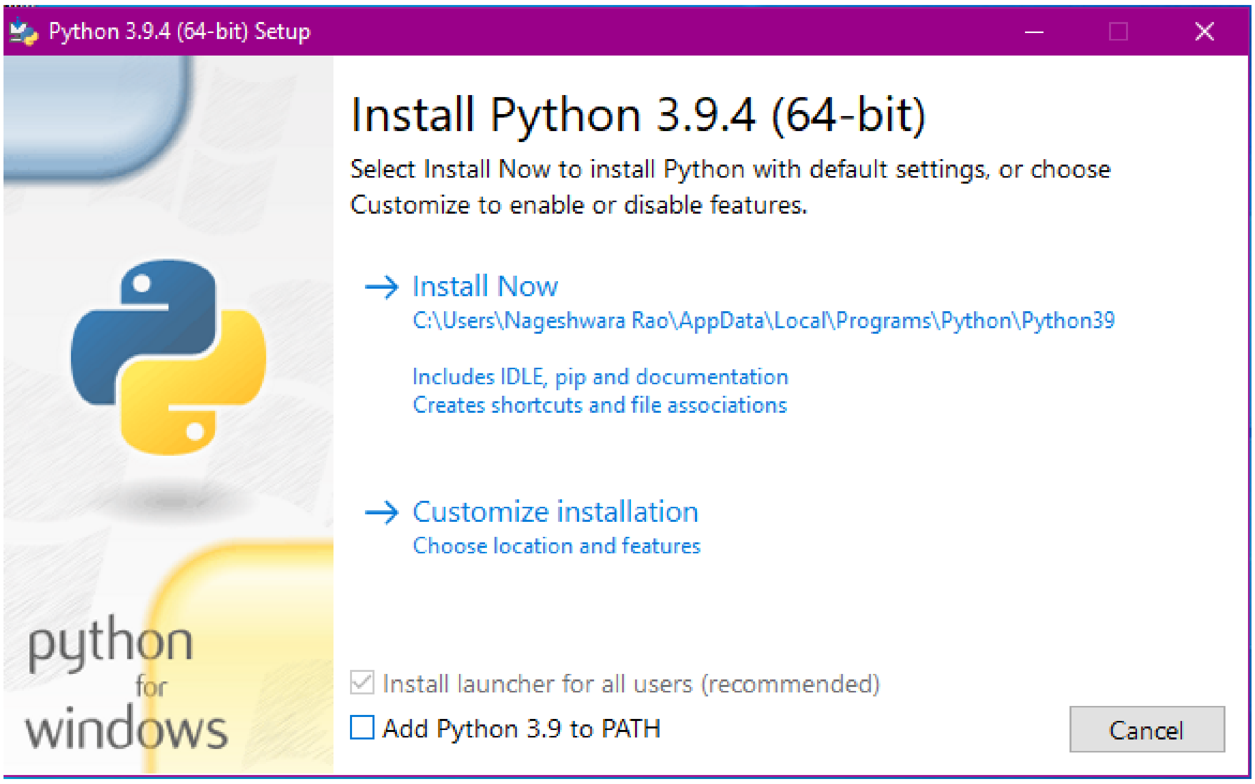
- Latest Python 3 Release - Python 3.9.4
- Latest Python 2 Release - Python 2.7.18

Stable Releases

- Python 3.9.4 - April 4, 2021
  Note that Python 3.9.4 *cannot* be used on Windows 7 or earlier.
  - Download Windows embeddable package (32-bit)
  - Download Windows embeddable package (64-bit)
  - Download Windows help file
  - Download Windows installer (32-bit)
  - Download Windows installer (64-bit)
- Python 3.9.3 - April 2, 2021
  Note that Python 3.9.3 *cannot* be used on Windows 7 or earlier.

Figure 1.5: Selecting Windows installer to install Python.

This will download Windows installer by the name 'python-3.9.4-amd64.exe' file into our computer. Double click on this file and the installer will run and display 'Setup' window. Go to the bottom of the screen and select the check box at 'Add Python 3.9 to PATH'. A tick mark will appear in that checkbox. Then come up and click on 'Install Now'. See Figure 1.6.


Python 3.9.4 (64-bit) Setup

Install Python 3.9.4 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

→ Install Now
C:\Users\Nageshwara Rao\AppData\Local\Programs\Python\Python39

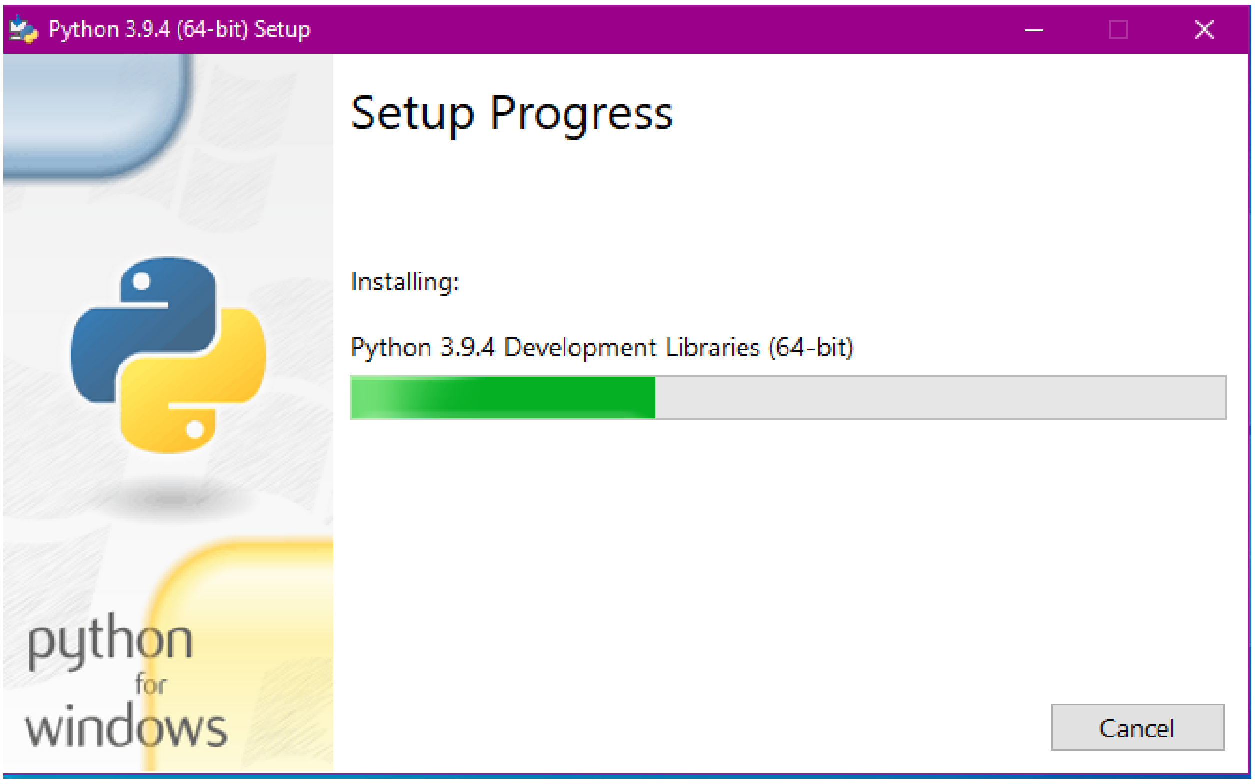
Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.9 to PATH

Figure 1.6: Starting screen of Installation of Python

Installation Progress bar will appear as shown in Figure 1.7 and we have to wait for a few minutes.


Python 3.9.4 (64-bit) Setup

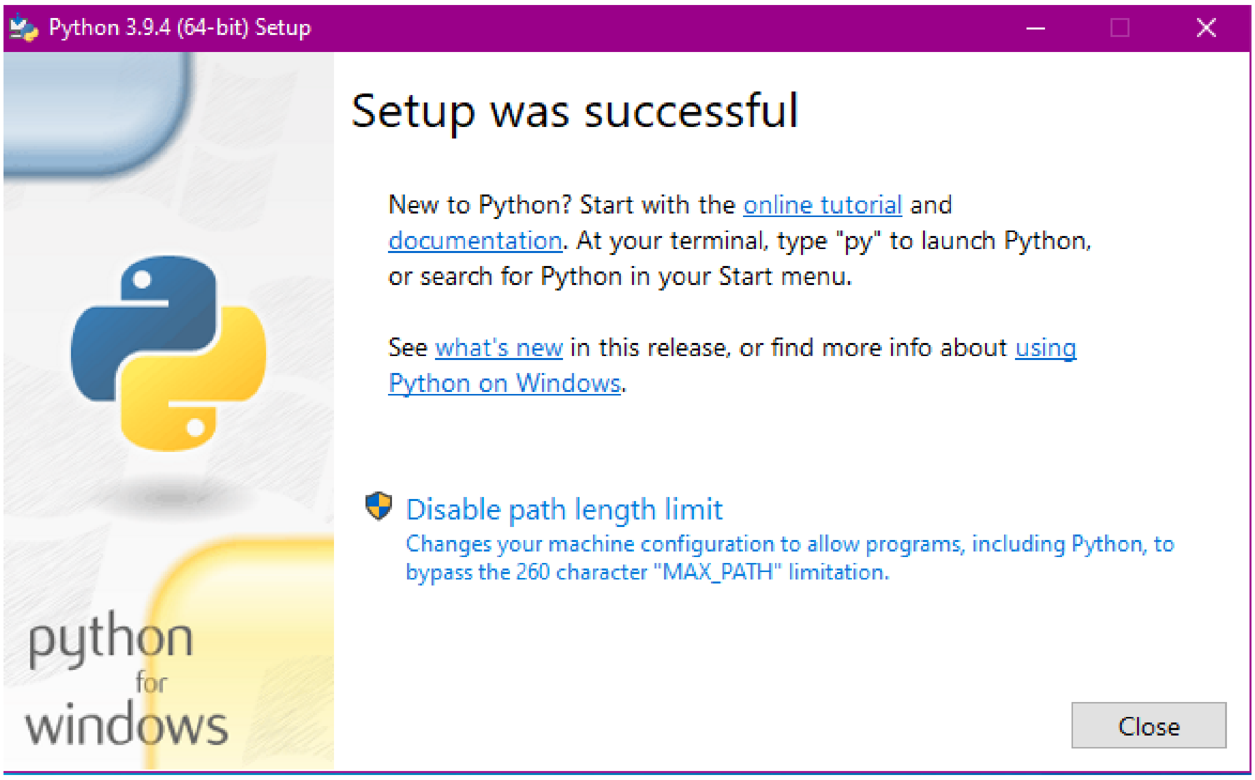
Setup Progress

Installing:

Python 3.9.4 Development Libraries (64-bit)

Figure 1.7: Installation of Python is going on.

After completion, ‘Setup was successful’ message can be seen. Click on ‘Close’ button as shown in Figure 1.8. This completes installation of Python in our System.



Setup was successful

New to Python? Start with the online tutorial and documentation. At your terminal, type "py" to launch Python, or search for Python in your Start menu.

See what's new in this release, or find more info about using Python on Windows.

Disable path length limit

Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

Figure 1.8: Installation of Python completed.

Installing additional packages

Along with basic Python software, we need to install additional packages developed by other companies. These additional packages are nothing but additional libraries (or code) which can be imported and used in our Python programs. We may require to install the following packages: numpy, pandas, xlrd, matplotlib and seaborn. These packages can be installed by going to DOS prompt (or System prompt or command prompt). To go to DOS prompt, we can Right click on Windows ‘Start’ button and then click on ‘Run’. Then type ‘cmd’ or ‘command’ to go to System prompt. This is shown in Figure 1.9.

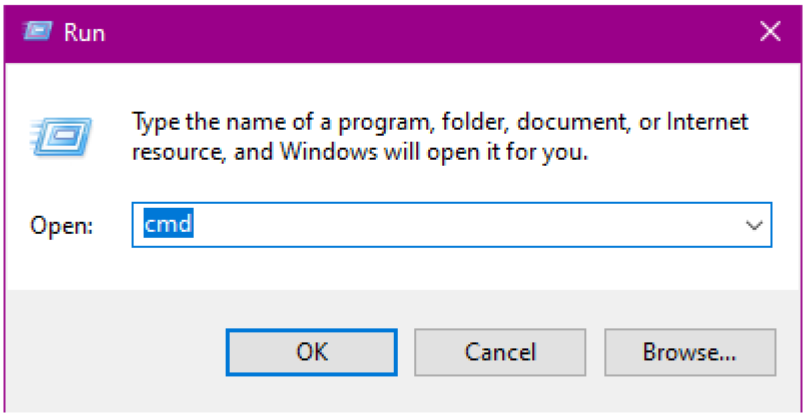

Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open:
cmd
OK
Cancel
Browse...

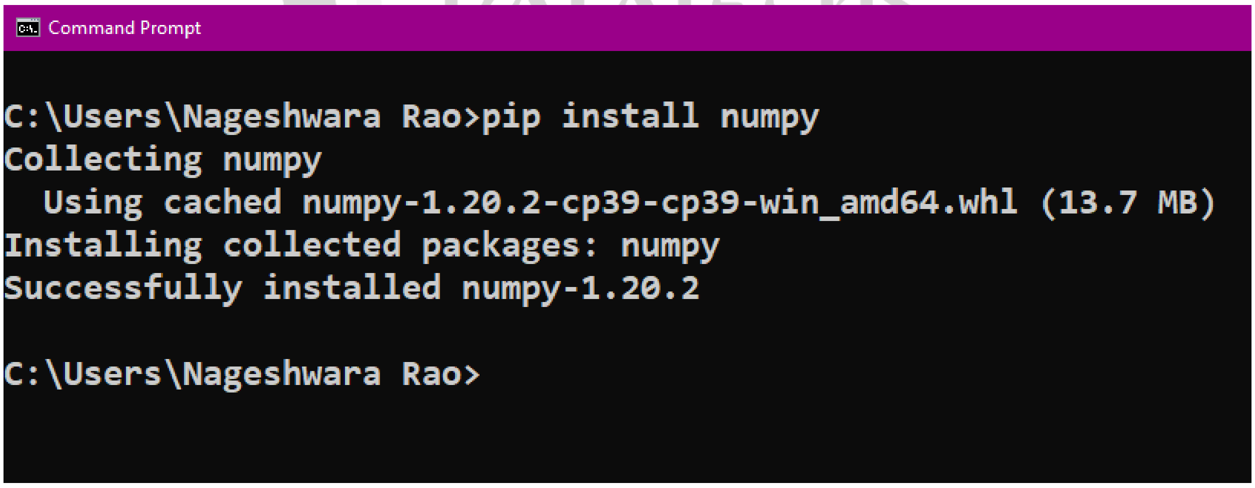
Figure 1.9: Going to DOS System prompt

Installing numpy

Python supports only single dimensional arrays. To create multi-dimensional arrays, we need a special package called Numerical Python package or *numpy*. To download and install this package, first we should go to System prompt and then use 'pip' (Python Installation of Packages) command as shown below:

```
      C:\> pip install numpy
```

'pip' program comes with Python software by default. When this command is given, it searches the latest version of the numpy package on the Internet, downloads it and installs it, as shown in Figure 1.10:



```
C:\Users\Nageshwara Rao>pip install numpy
Collecting numpy
  Using cached numpy-1.20.2-cp39-cp39-win_amd64.whl (13.7 MB)
Installing collected packages: numpy
Successfully installed numpy-1.20.2

C:\Users\Nageshwara Rao>
```

Figure 1.10: Installation of numpy package

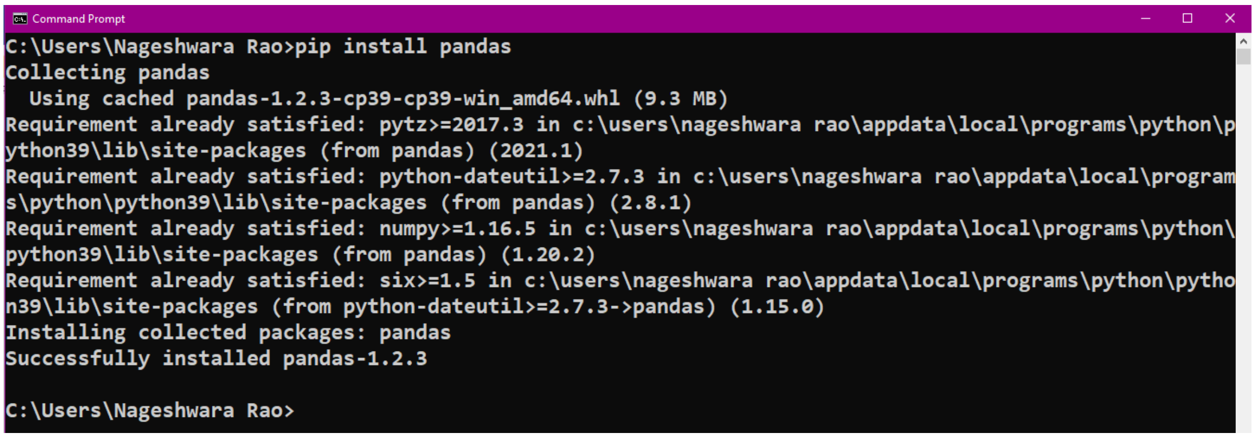
Of course, our computer would have been connected to the Internet while using this command.

Installing pandas

pandas is a package used in data analysis. This package is mostly used by data scientists and data analysts. To download and install this package, we should go to System prompt and then use 'pip' (Python Installation of Packages) command as shown below:

```
      C:\> pip install pandas
```

This command downloads pandas package from the Internet and installs it, as shown in Figure 1.11:



```
C:\Users\Nageshwara Rao>pip install pandas
Collecting pandas
  Using cached pandas-1.2.3-cp39-cp39-win_amd64.whl (9.3 MB)
Requirement already satisfied: pytz>=2017.3 in c:\users\nageshwara rao\appdata\local\programs\python\p
ython39\lib\site-packages (from pandas) (2021.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\nageshwara rao\appdata\local\program
s\python\python39\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: numpy>=1.16.5 in c:\users\nageshwara rao\appdata\local\programs\python\
python39\lib\site-packages (from pandas) (1.20.2)
Requirement already satisfied: six>=1.5 in c:\users\nageshwara rao\appdata\local\programs\python\pytho
n39\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Installing collected packages: pandas
Successfully installed pandas-1.2.3

C:\Users\Nageshwara Rao>
```

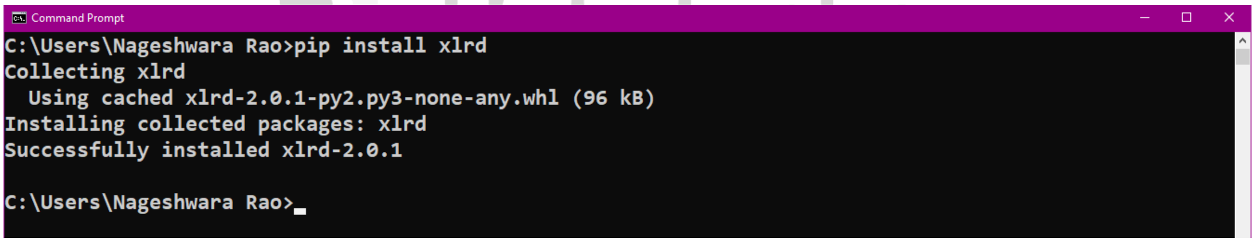
Figure 1.11: Installation of pandas package

Installing xlrd

xlrd is a package that is useful to retrieve data from Microsoft Excel spreadsheet files. Hence, it is useful in data analysis. To download and install this package, we should go to System prompt and then use 'pip' (Python Installation of Packages) command as shown below:

```
        C:\> pip install xlrd
```

This command downloads xlrd package from the Internet and installs it, as shown in Figure 1.12:



```
C:\Users\Nageshwara Rao>pip install xlrd
Collecting xlrd
  Using cached xlrd-2.0.1-py2.py3-none-any.whl (96 kB)
Installing collected packages: xlrd
Successfully installed xlrd-2.0.1

C:\Users\Nageshwara Rao>_
```

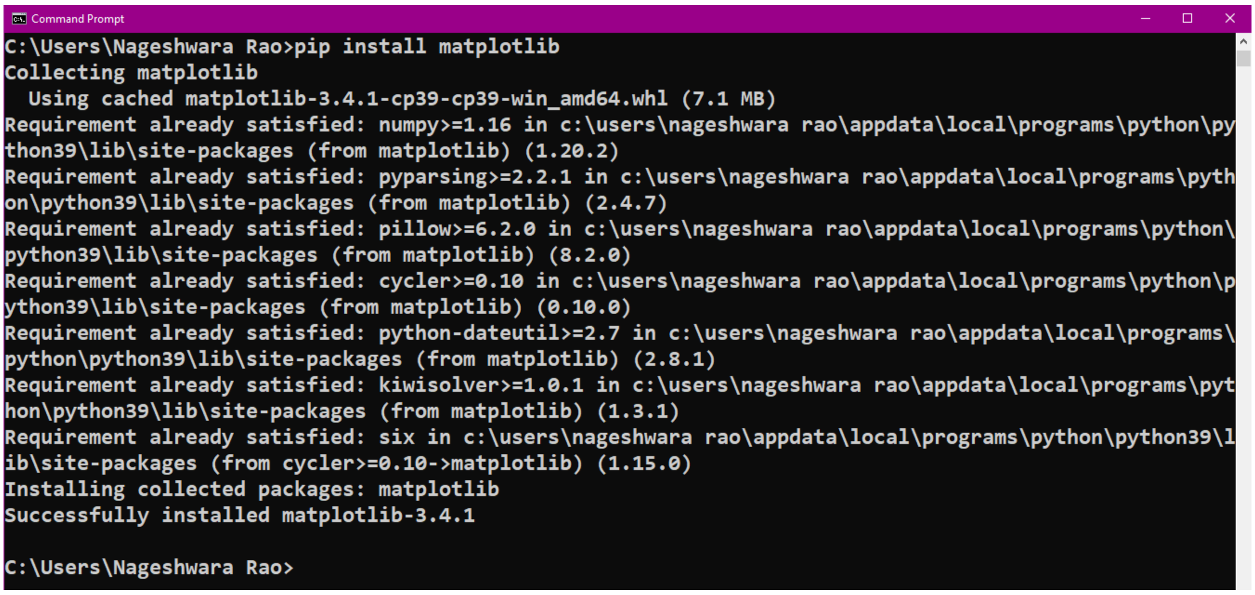
Figure 1.12: Installation of xlrd package

Installing matplotlib

matplotlib is another important package in Python that is useful to produce good quality 2D graphics. It is mostly used for the purpose of showing data in the form of graphs and also for designing electronic circuits, machinery parts, etc. To download and install this package, we should go to System prompt and then use 'pip' (Python Installation of Packages) command as shown below:

```
        C:\> pip install matplotlib
```

This command downloads the matplotlib package from the Internet and installs it as shown in Figure 1.13:



```
C:\Users\Nageshwara Rao>pip install matplotlib
Collecting matplotlib
  Using cached matplotlib-3.4.1-cp39-cp39-win_amd64.whl (7.1 MB)
Requirement already satisfied: numpy>=1.16 in c:\users\nageshwara rao\appdata\local\programs\python\py
thon39\lib\site-packages (from matplotlib) (1.20.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\nageshwara rao\appdata\local\programs\pyth
on\python39\lib\site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\nageshwara rao\appdata\local\programs\python\
python39\lib\site-packages (from matplotlib) (8.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\nageshwara rao\appdata\local\programs\python\p
ython39\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\nageshwara rao\appdata\local\programs\
python\python39\lib\site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\nageshwara rao\appdata\local\programs\pyt
hon\python39\lib\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: six in c:\users\nageshwara rao\appdata\local\programs\python\python39\l
ib\site-packages (from cycler>=0.10->matplotlib) (1.15.0)
Installing collected packages: matplotlib
Successfully installed matplotlib-3.4.1

C:\Users\Nageshwara Rao>
```

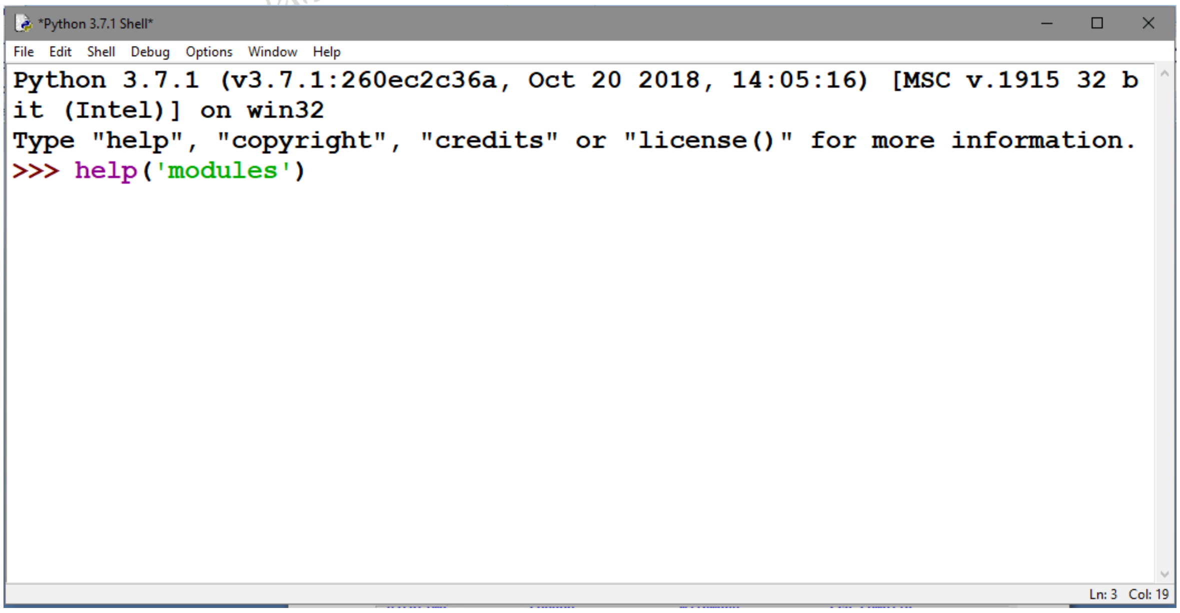
Figure 1.13: Installation of matplotlib package

Verifying Installed Packages

We can verify whether the installed packages are added to our Python software properly or not by going into Python and typing the following command at Python prompt (triple greater than symbol) as:

>>> help('modules')

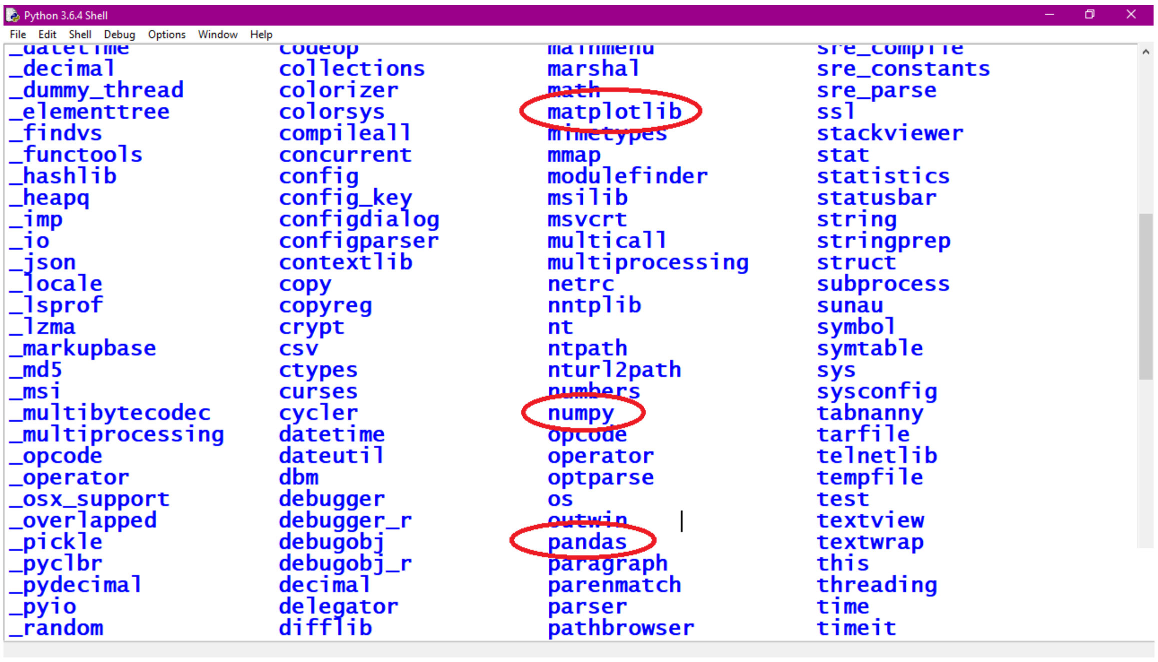
For this purpose, first click on the Python IDLE Window pinned at the taskbar and then type the preceding command as shown in the Figure 2.12.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 b
it (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> help('modules')
```

Figure 2.12: To view the modules available in Python

It will display all the module names available currently in your Python software as shown in the Figure 2.13. You can verify that your Python software now has the packages like: numpy, xlrd, pandas, and matplotlib available as modules.

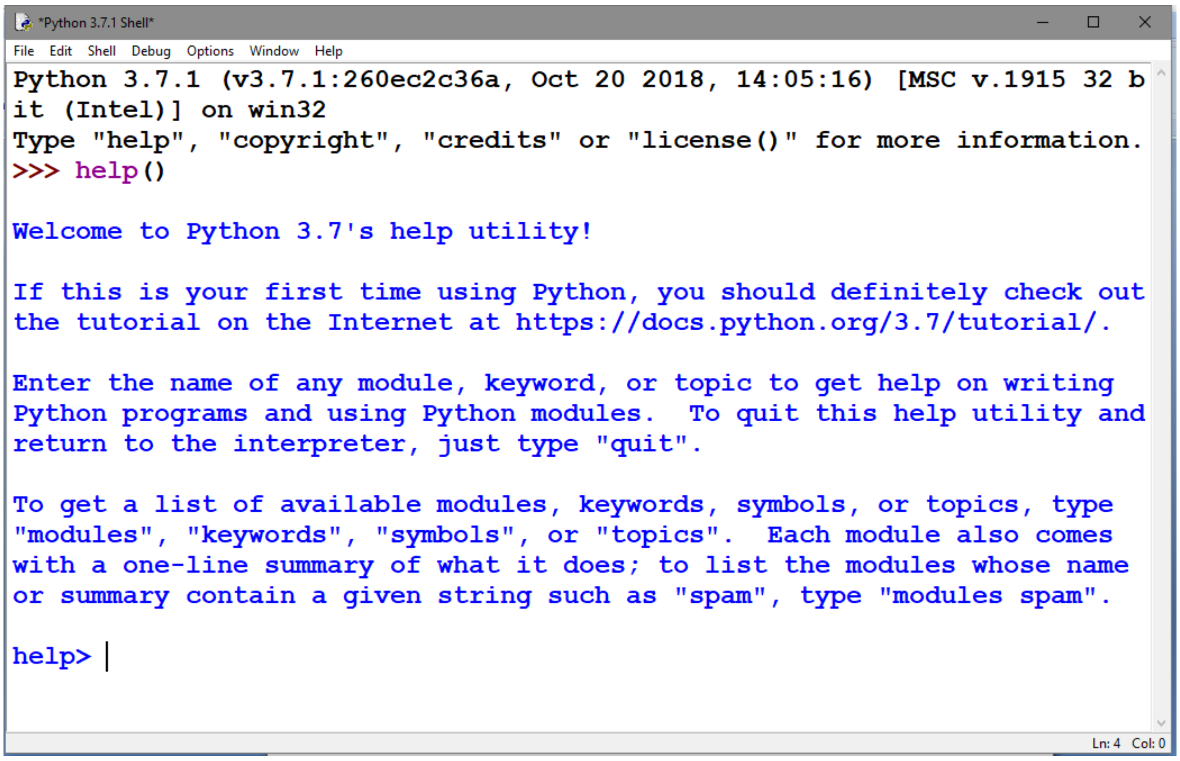


```
_datetime           codeop              mailmenu            sre_compile
_decimal            collections         marshal             sre_constants
_dummy_thread       colorizer           math                sre_parse
_elementtree        colorsys            matplotlib          ssl
_findvs             compileall          mimetypes           stackviewer
_functools          concurrent          mmap                stat
_hashlib            config              modulefinder        statistics
_heapq              config_key          msilib              statusbar
_imp                configdialog        msvcrt              string
_io                 configparser        multicall           stringprep
_json               contextlib          multiprocessing     struct
_locale             copy                netrc               subprocess
_lsprof             copyreg             nntplib             sunau
_lzma               crypt               nt                  symbol
_markupbase         csv                 ntpath              symtable
_md5                ctypes              nturl2path          sys
_msi                curses              numbers             sysconfig
_multibytecodec     cycler              numpy               tabnanny
_multiprocessing    datetime            opcode              tarfile
_opcode             dateutil            operator            telnetlib
_operator           dbm                 optparse            tempfile
_osx_support        debugger            os                  test
_overlapped         debugger_r          outwin              textview
_pickle             debugobj            pandas              textwrap
_pyclbr             debugobj_r          paragraph           this
_pydecimal          decimal             parenmatch          threading
_pyio               delegator           parser              time
_random             difflib             pathbrowser         timeit
```

Figure 2.13: To verify installed packages in Python

Getting Help in Python

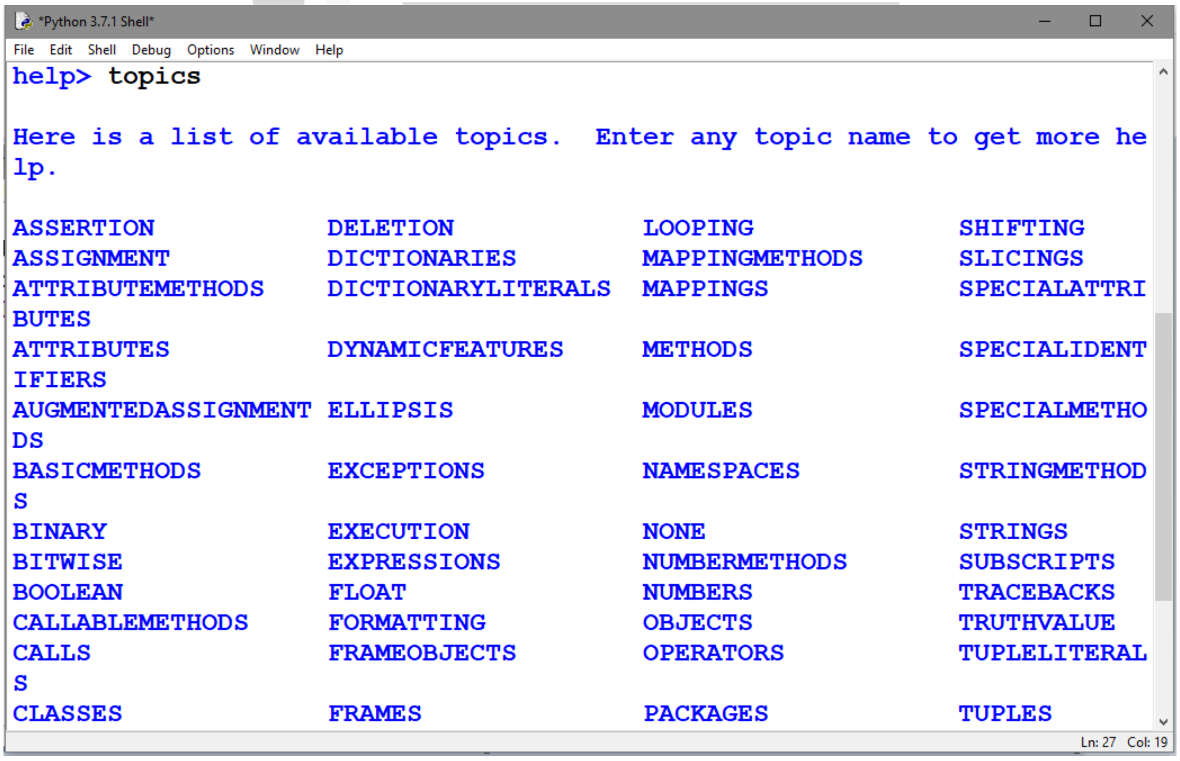
When a programmer faces some doubt about how to use a particular feature of the Python language, he can view the help. To get help, one can enter help mode by typing help() at Python prompt (i.e. >>> prompt). We can see the help utility appearing as shown in Figure 2.14.



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 b
it (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> help()

Welcome to Python 3.7's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.7/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help> |
```

Figure 2.14: Entering Help Mode in Python

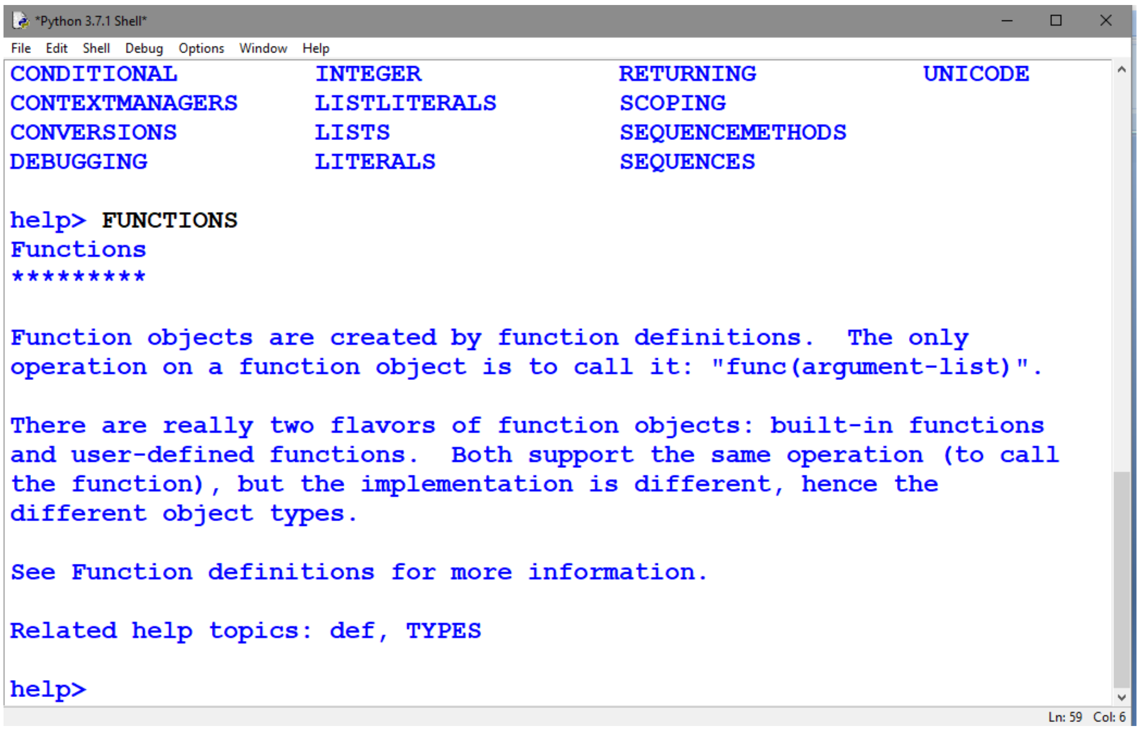
Now, we can type 'modules' to see which modules are available in Python. We can type 'topics' to know about topics in Python. Let's enter topics at the help prompt, as shown in Figure 2.15.



```
help> topics

Here is a list of available topics.  Enter any topic name to get more he
lp.

ASSERTION           DELETION            LOOPING             SHIFTING
ASSIGNMENT          DICTIONARIES        MAPPINGMETHODS      SLICINGS
ATTRIBUTEMETHODS    DICTIONARYLITERALS  MAPPINGS            SPECIALATTRI
BUTES
ATTRIBUTES          DYNAMICFEATURES     METHODS             SPECIALIDENT
IFIERS
AUGMENTEDASSIGNMENT ELLIPSIS            MODULES             SPECIALMETHO
DS
BASICMETHODS        EXCEPTIONS          NAMESPACES          STRINGMETHOD
S
BINARY              EXECUTION           NONE                STRINGS
BITWISE             EXPRESSIONS         NUMBERMETHODS       SUBSCRIPTS
BOOLEAN             FLOAT               NUMBERS             TRACEBACKS
CALLABLEMETHODS     FORMATTING          OBJECTS             TRUTHVALUE
CALLS               FRAMEOBJECTS        OPERATORS           TUPLELITERAL
S
CLASSES             FRAMES              PACKAGES            TUPLES
```

Figure 2.15: Getting Help on Topics

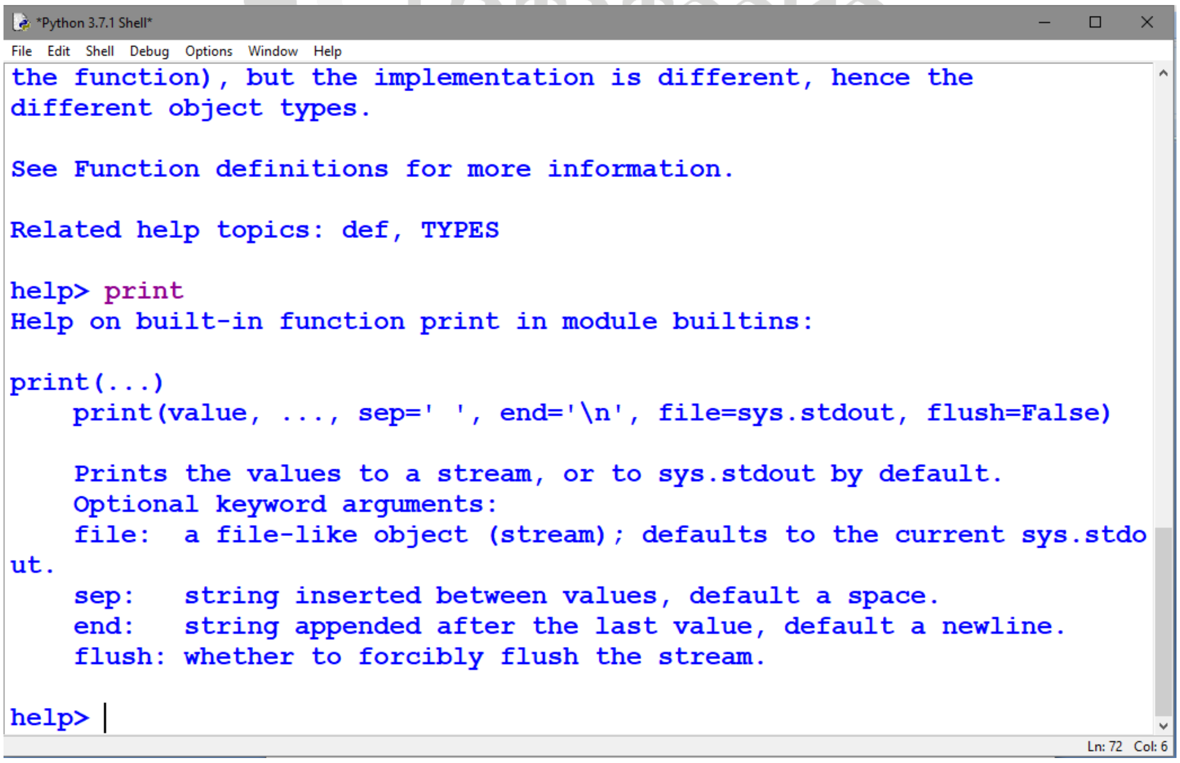
In the topics, suppose we want to know about functions, we should enter 'FUNCTIONS' in capital letters since the FUNCTIONS topic is shown in capital letters in the help window, as shown in Figure 2.16.



```
CONDITIONAL          INTEGER              RETURNING            UNICODE
CONTEXTMANAGERS      LISTLITERALS         SCOPING
CONVERSIONS          LISTS                SEQUENCEMETHODS
DEBUGGING            LITERALS             SEQUENCES

help> FUNCTIONS
Functions
*********

Function objects are created by function definitions.  The only
operation on a function object is to call it: "func(argument-list)".

There are really two flavors of function objects: built-in functions
and user-defined functions.  Both support the same operation (to call
the function), but the implementation is different, hence the
different object types.

See Function definitions for more information.

Related help topics: def, TYPES

help>
```

Figure 2.16: Getting Help on Functions

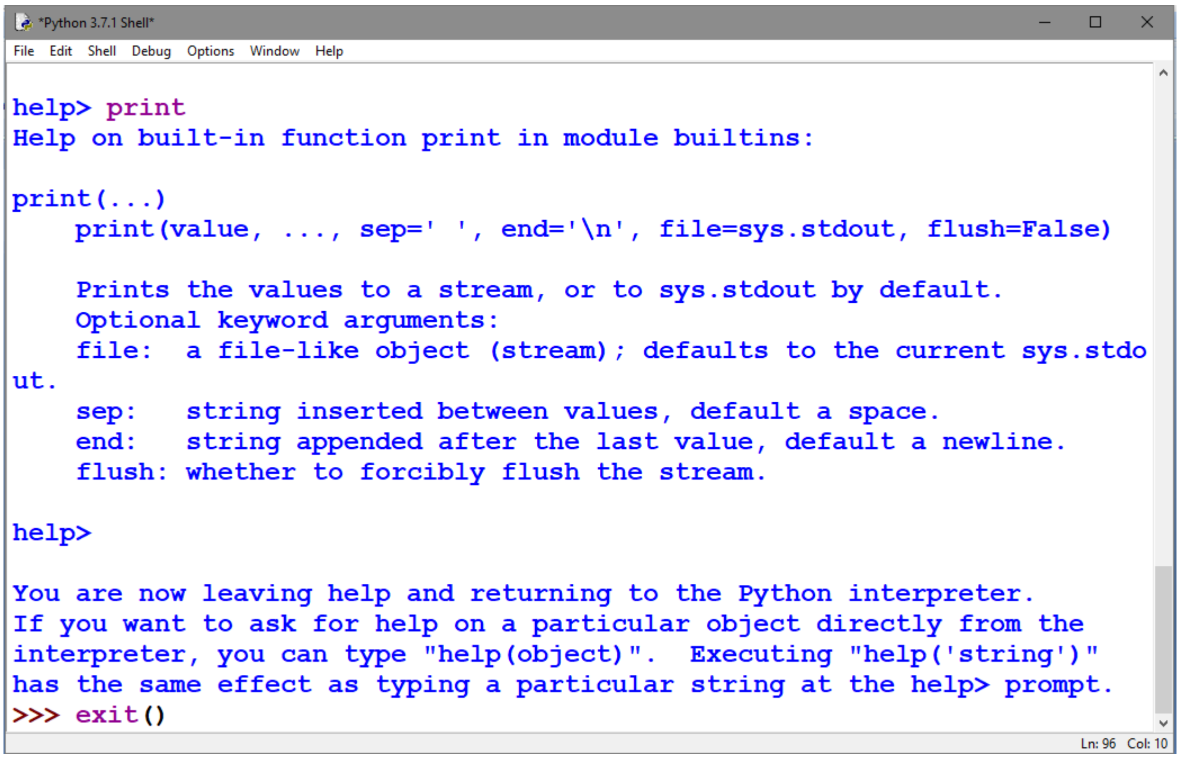
To get help on the print function, we can type 'print' at the help prompt, as shown in Figure 2.17.



```
the function), but the implementation is different, hence the
different object types.

See Function definitions for more information.

Related help topics: def, TYPES

help> print
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdo
ut.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

help> |
```

Figure 2.17: Getting Help on Print Function

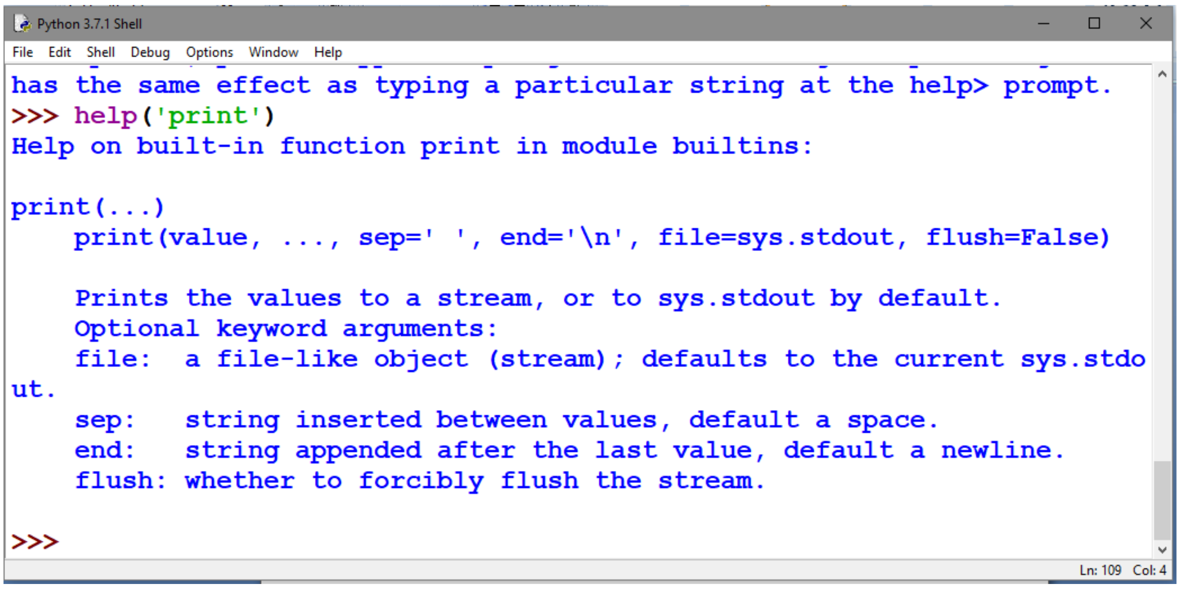
To quit from the help mode, we should simply press the Enter button once again without typing anything. A message appears that we are leaving the help mode and then we arrive at the Python prompt, i.e. >>>. To exit the Python interpreter (or PVM), we should type either exit() or quit(), as shown in Figure 2.18.



```
help> print
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdo
ut.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

help>

You are now leaving help and returning to the Python interpreter.
If you want to ask for help on a particular object directly from the
interpreter, you can type "help(object)".  Executing "help('string')"
has the same effect as typing a particular string at the help> prompt.
>>> exit()
```

Figure 2.18: Exiting the Python IDLE Window

We can also view help without entering the help mode. Viewing help is possible at the Python prompt. We can use the help() command and inside the parentheses, type the item name in single quotes. For example, to get help on topics, we can type help (‘topics’) and to get help on the print function, we can type help(‘print’), as shown in Figure 2.19:

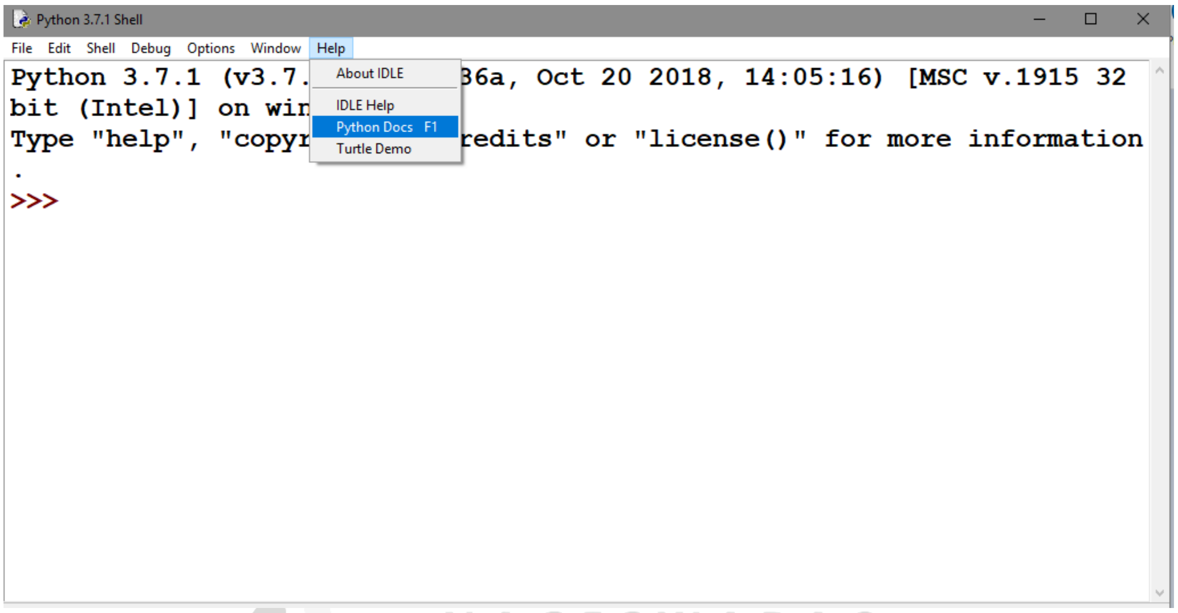


```
has the same effect as typing a particular string at the help> prompt.
>>> help('print')
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdo
ut.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.

>>>
```

Figure 2.19: Using the help() function

Getting Python Documentation Help

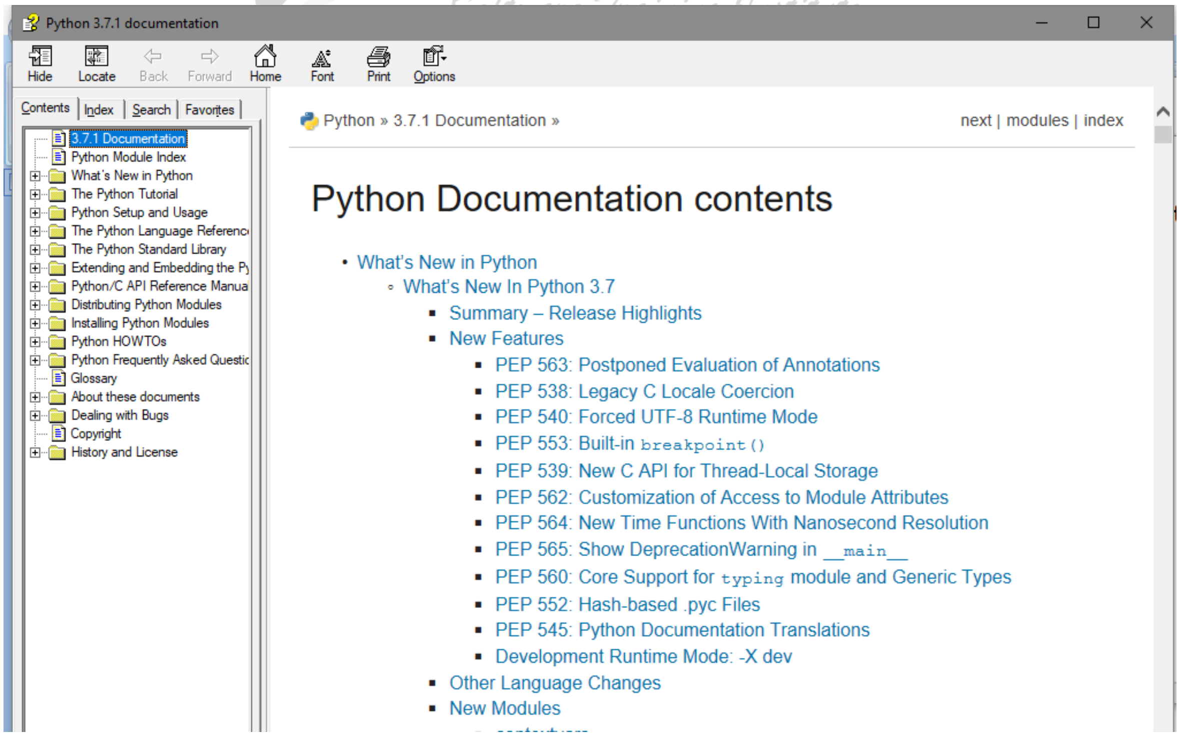
Python developers have provided extensive description of all Python features in a document that is called ‘Python documents’. This provides a great help for beginners and for professional programmers to understand all the features of Python. To see Python documentation help, we should open the IDLE window. Then select Help → Python Docs, as shown in Figure 2.20:



```
Python 3.7.1 (v3.7.  36a, Oct 20 2018, 14:05:16) [MSC v.1915 32
bit (Intel)] on wir
Type "help", "copyr  redits" or "license()" for more information
.
>>>
```

Figure 2.20: Getting Help from Python Documentation

It will display the Python 3.7.1 documentation help with all available features so that one can select any feature by clicking on it, as shown in Figure 2.21:



Python Documentation contents

- What's New in Python
  - What's New In Python 3.7
    - Summary – Release Highlights
    - New Features
      - PEP 563: Postponed Evaluation of Annotations
      - PEP 538: Legacy C Locale Coercion
      - PEP 540: Forced UTF-8 Runtime Mode
      - PEP 553: Built-in `breakpoint()`
      - PEP 539: New C API for Thread-Local Storage
      - PEP 562: Customization of Access to Module Attributes
      - PEP 564: New Time Functions With Nanosecond Resolution
      - PEP 565: Show DeprecationWarning in `__main__`
      - PEP 560: Core Support for `typing` module and Generic Types
      - PEP 552: Hash-based .pyc Files
      - PEP 545: Python Documentation Translations
      - Development Runtime Mode: -X dev
    - Other Language Changes
    - New Modules

Figure 2.21: The Python documentation window

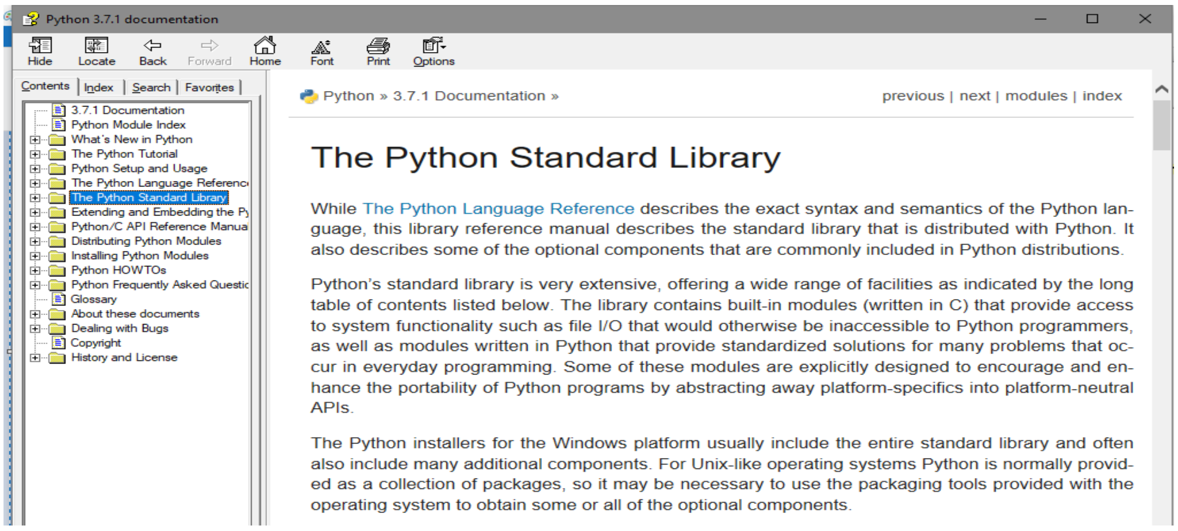
At the left side frame, we see contents out of which the following are very useful for us:

The Python Tutorial

The Python Language Reference

The Python Standard Library

For example, click 'The Python Standard Library' to see the topic-wise comprehensive help on Python Standard Library. We can click on the close button (X) to close this help window, as shown in Figure 2.22.


Python 3.7.1 documentation
Hide
Locate
Back
Forward
Home
Font
Print
Options
Contents
Index
Search
Favorites
3.7.1 Documentation
Python Module Index
What's New in Python
The Python Tutorial
Python Setup and Usage
The Python Language Referenc
The Python Standard Library
Extending and Embedding the Py
Python/C API Reference Manua
Distributing Python Modules
Installing Python Modules
Python HOWTOs
Python Frequently Asked Questic
Glossary
About these documents
Dealing with Bugs
Copyright
History and License
Python » 3.7.1 Documentation »
previous | next | modules | index

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

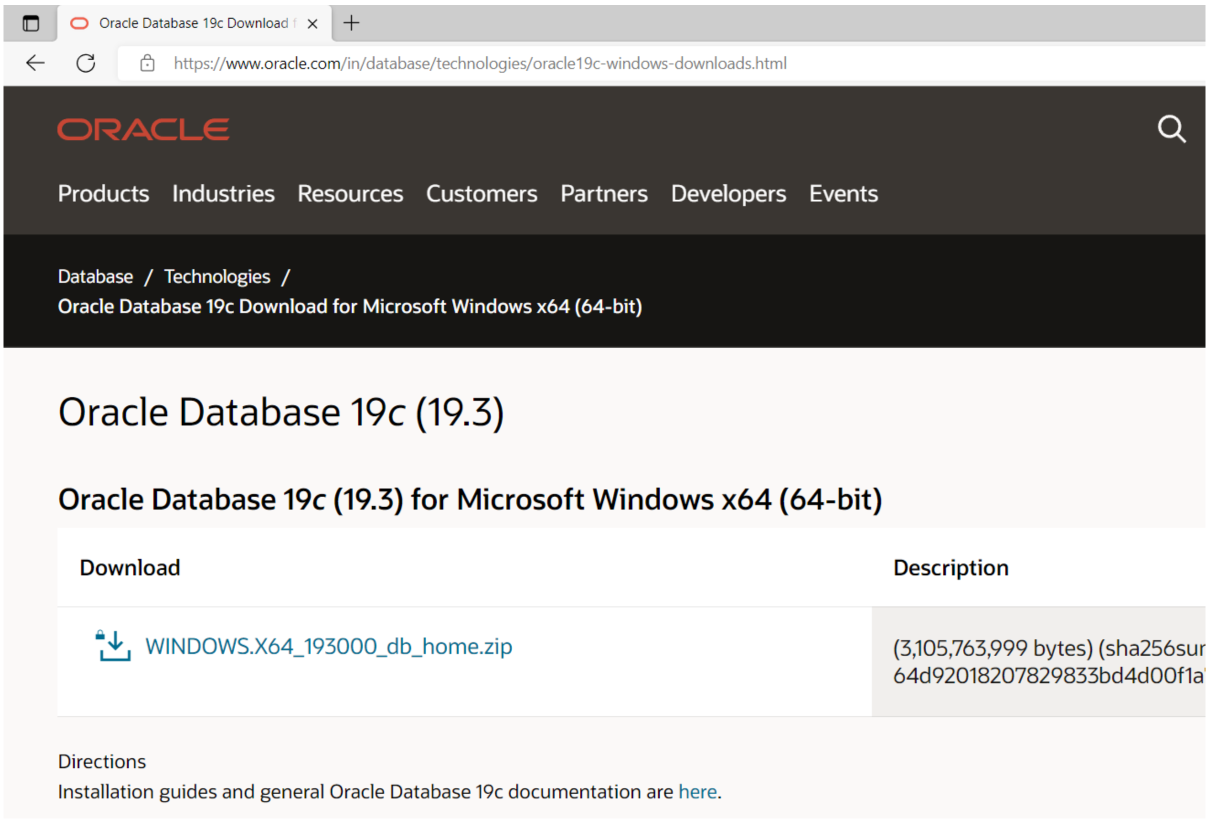
Figure 2.22: Going to Python Standard Library

INSTALLATION OF DATABASES AND DRIVERS

ORACLE 19C INSTALLATION

1. First of all, we should visit oracle.com website and in that site, the following page:

https://www.oracle.com/in/database/technologies/oracle19c-windows-downloads.html


Oracle Database 19c Download f

https://www.oracle.com/in/database/technologies/oracle19c-windows-downloads.html

ORACLE
Products Industries Resources Customers Partners Developers Events

Database / Technologies /
Oracle Database 19c Download for Microsoft Windows x64 (64-bit)

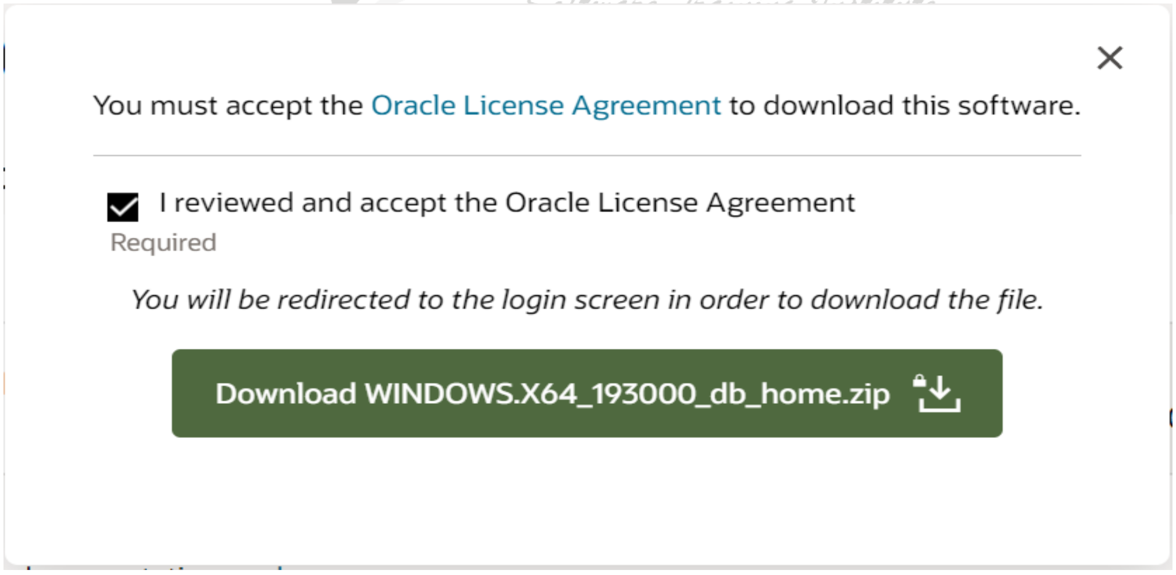
Oracle Database 19c (19.3)

Oracle Database 19c (19.3) for Microsoft Windows x64 (64-bit)

Download	Description
WINDOWS.X64_193000_db_home.zip	(3,105,763,999 bytes) (sha256sur 64d92018207829833bd4d00f1a

Directions
Installation guides and general Oracle Database 19c documentation are here.

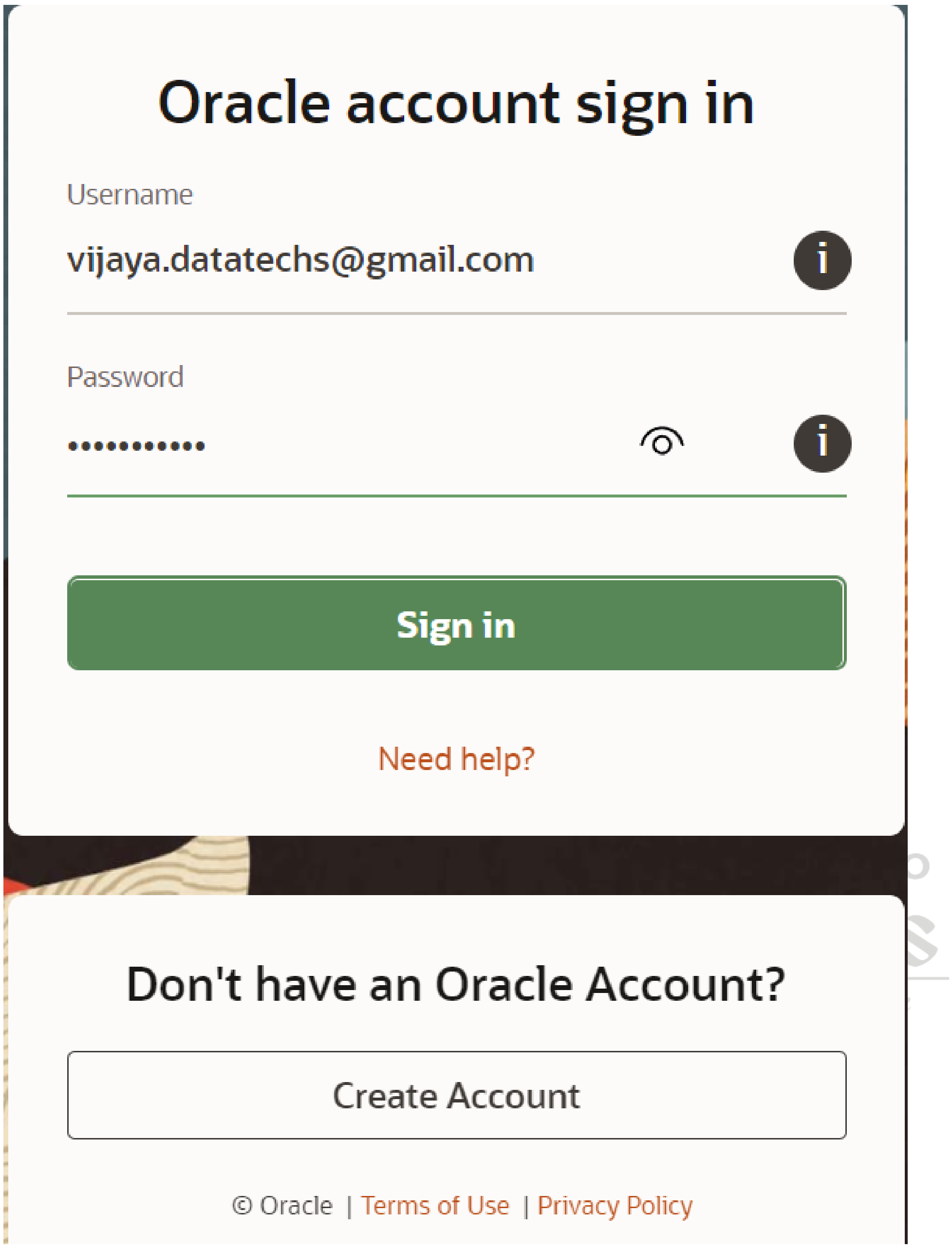
2. Click on ‘WINDOWS.X64_193000_db_home.zip’ file to download. In the next window, accept the Oracle License Agreement and then click on the button ‘Download WINDOWS.X64_193000_db_home.zip.


You must accept the Oracle License Agreement to download this software.

☑ I reviewed and accept the Oracle License Agreement
Required

You will be redirected to the login screen in order to download the file.

3. It will ask for Oracle account sign in. If you have an account in Oracle.com, then you can fill in your Username and Password, otherwise you can create an account by supplying your name, phone number, mail id etc. details.


Oracle account sign in

Username

vijaya.datatechs@gmail.com

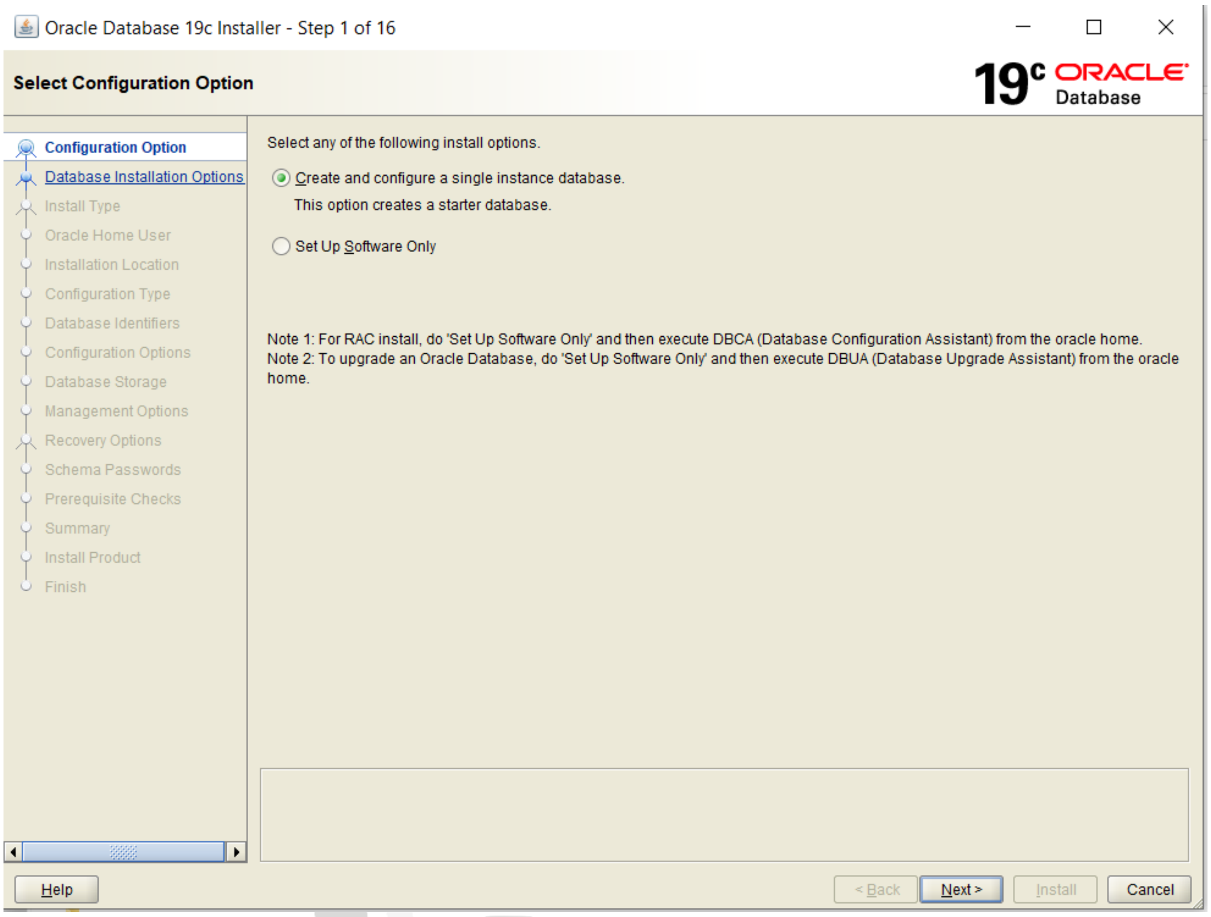
Password

••••••••••••

Sign in
Need help?
Don't have an Oracle Account?
Create Account
© Oracle | Terms of Use | Privacy Policy

4. The file will download into our system. We should unzip it using WinRar or WinZip softwares. It will create a separate folder by the name: WINDOWS.X64_193000_db_home. Go into this folder to find 'setup.exe'. Double click on it to start the installation.

5. In the first step, select 'Create and configure a single instance database' radio button and click on 'Next' button.



Select Configuration Option

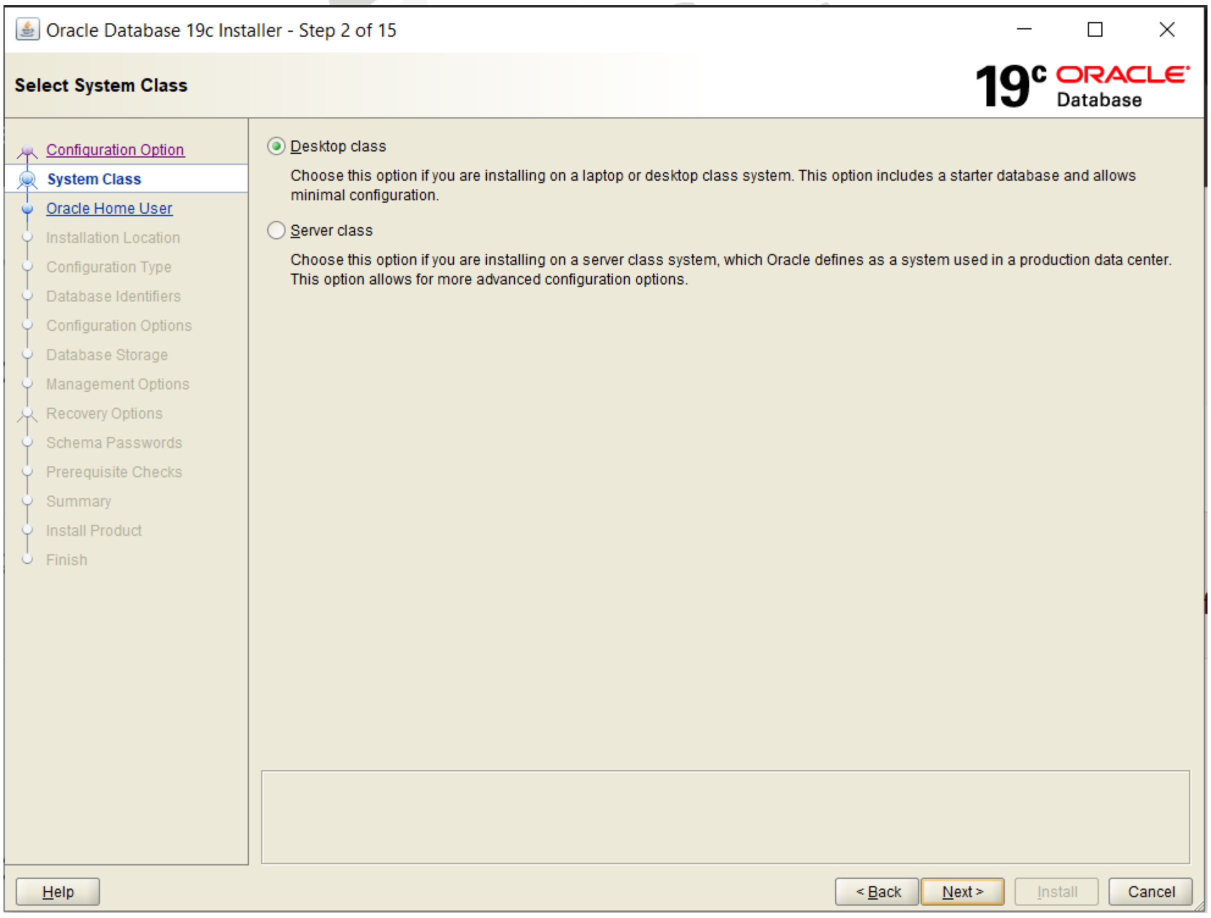
Select any of the following install options.

[✓ Create and configure a single instance database.]
This option creates a starter database.

Set Up Software Only

Note 1: For RAC install, do 'Set Up Software Only' and then execute DBCA (Database Configuration Assistant) from the oracle home.
Note 2: To upgrade an Oracle Database, do 'Set Up Software Only' and then execute DBUA (Database Upgrade Assistant) from the oracle home.

6. In the next screen, select 'Desktop class' radio button and click on 'Next' button.

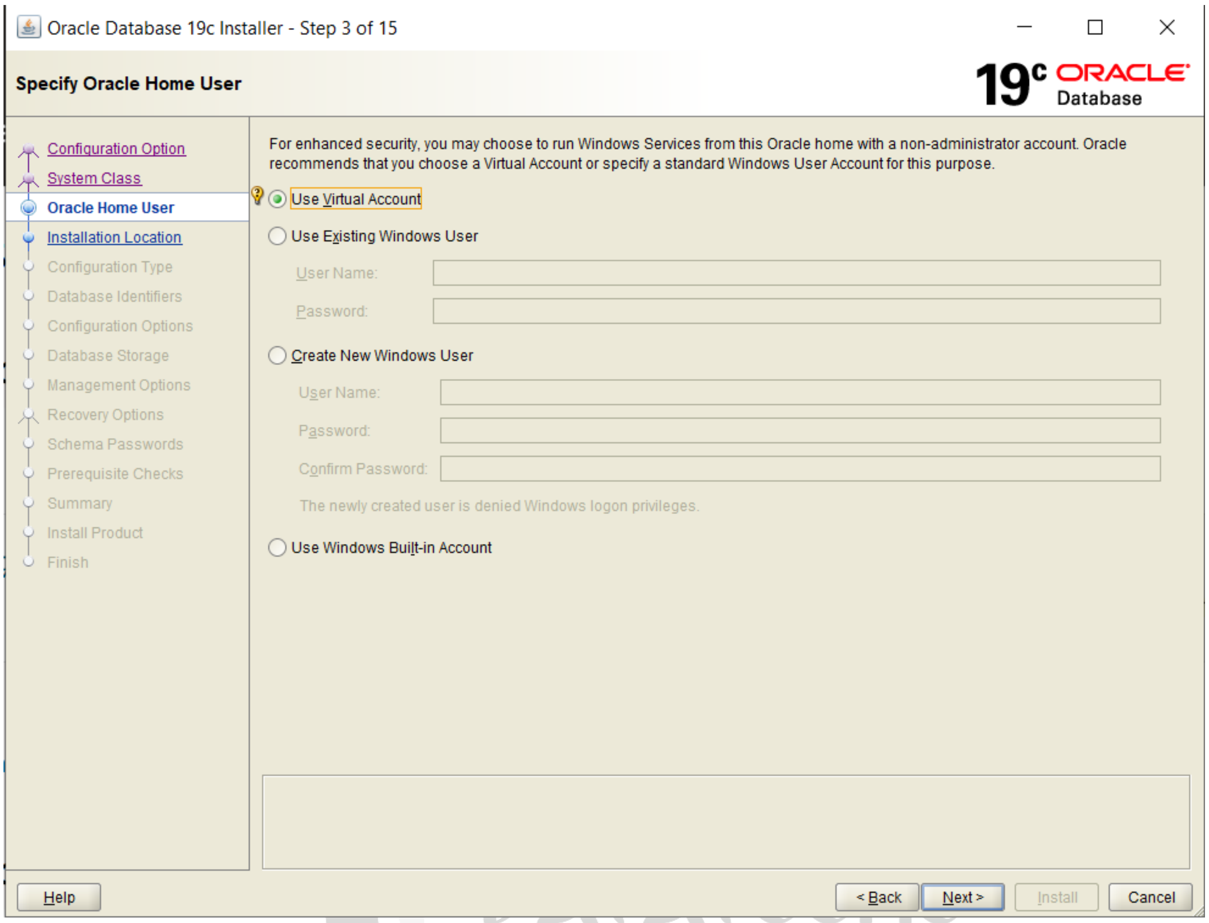


Select System Class

[✓ Desktop class]
Choose this option if you are installing on a laptop or desktop class system. This option includes a starter database and allows minimal configuration.

Server class
Choose this option if you are installing on a server class system, which Oracle defines as a system used in a production data center. This option allows for more advanced configuration options.

7. In the next screen, select ‘Use Virtual Account’ and click on ‘Next’ button.


Oracle Database 19c Installer - Step 3 of 15

Specify Oracle Home User

For enhanced security, you may choose to run Windows Services from this Oracle home with a non-administrator account. Oracle recommends that you choose a Virtual Account or specify a standard Windows User Account for this purpose.

[✓ Use Virtual Account]

Use Existing Windows User

User Name:

Password:

Create New Windows User

User Name:

Password:

Confirm Password:

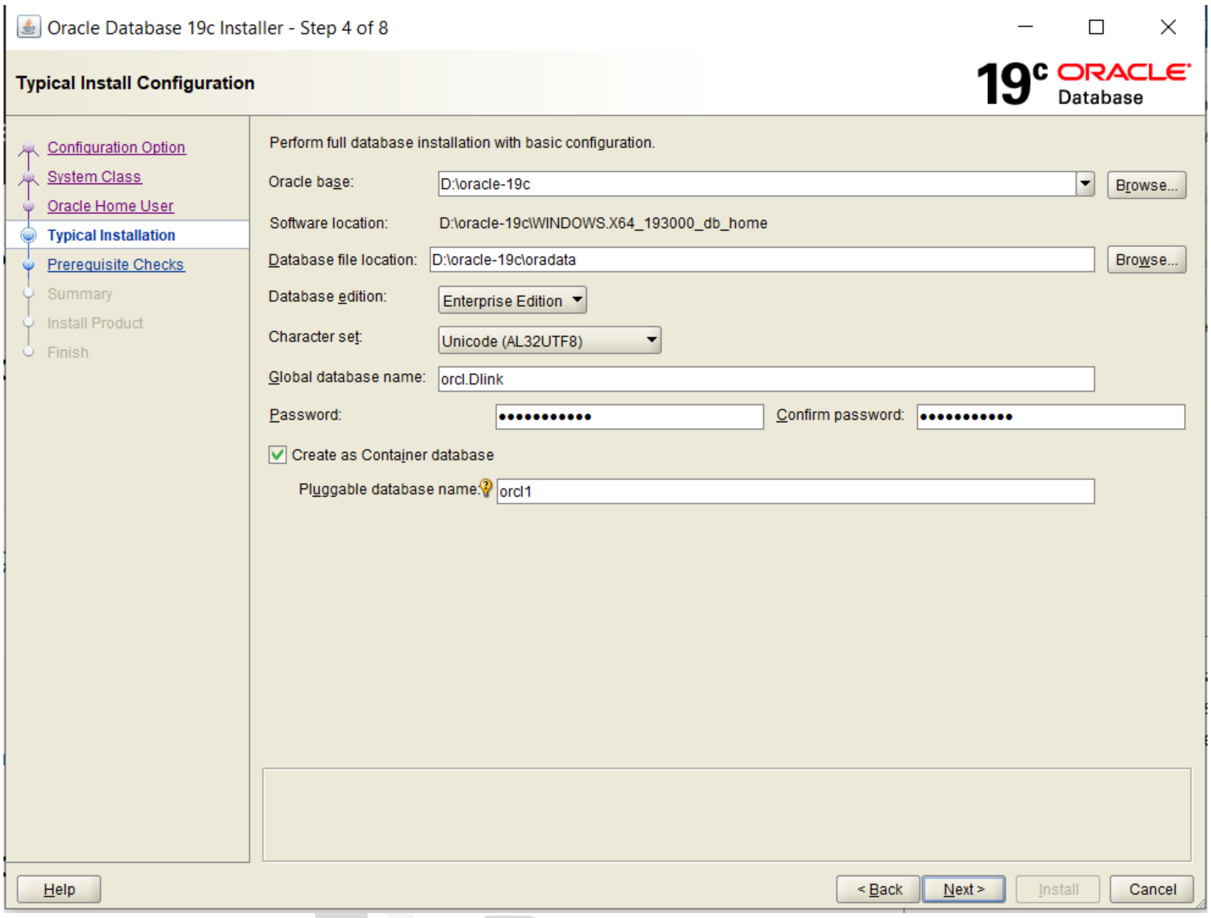
The newly created user is denied Windows logon privileges.

Use Windows Built-in Account

8. In the next page, type Password and confirm it. Then type Pluggable database name also. It shows some global database name that can be changed as we like. You have to remember this password and global database name. These are used in Python programs later to connect to oracle database.

NOTE: We typed Password: rnr_Oracle2

Global database name: orcl.Dlink (as shown by it)



Typical Install Configuration

Perform full database installation with basic configuration.

Oracle base: D:\oracle-19c

Software location: D:\oracle-19c\WINDOWS.X64_193000_db_home

Database file location: D:\oracle-19c\oradata

Database edition: Enterprise Edition

Character set: Unicode (AL32UTF8)

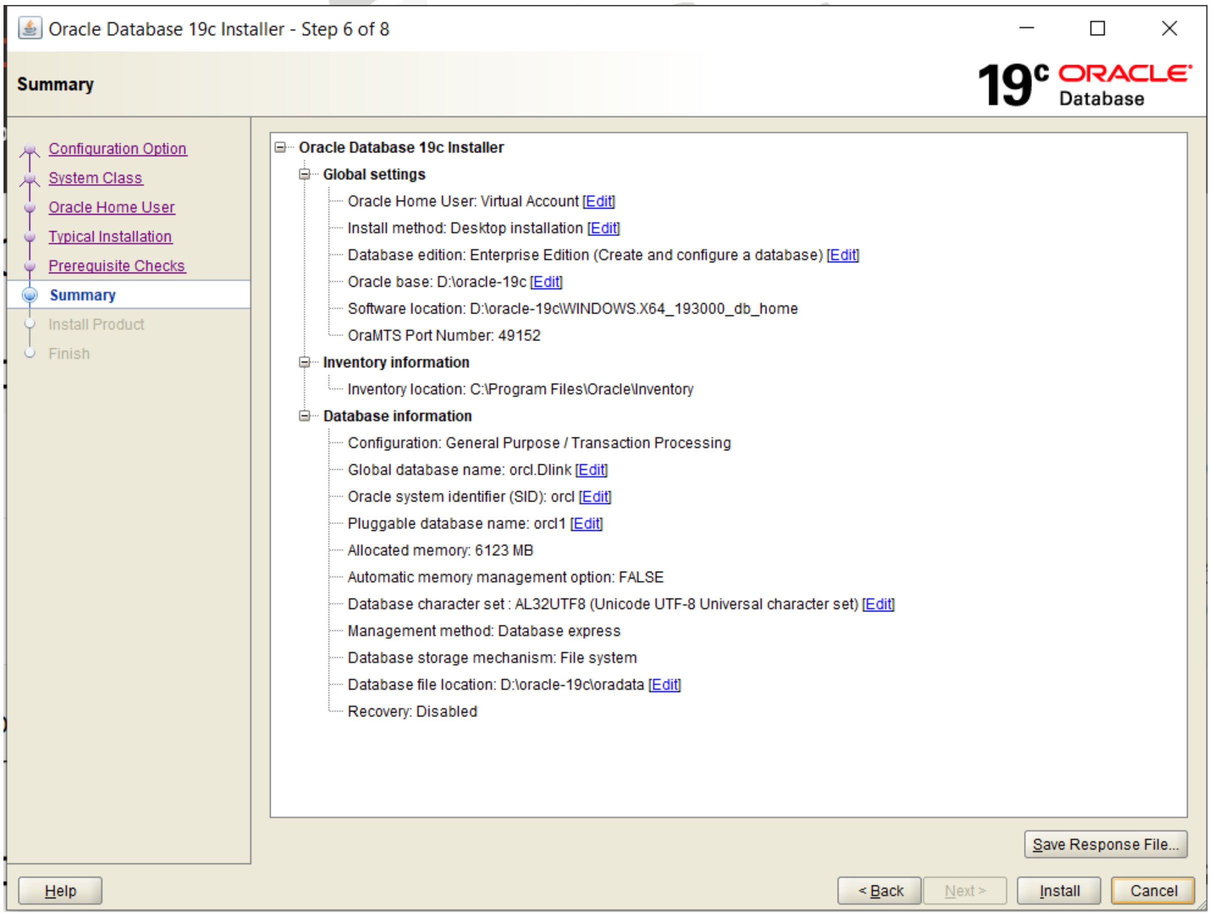
Global database name: orcl.Dlink

Password: ●●●●●●●●●●● Confirm password: ●●●●●●●●●●●

[✓] Create as Container database

Pluggable database name: orcl1

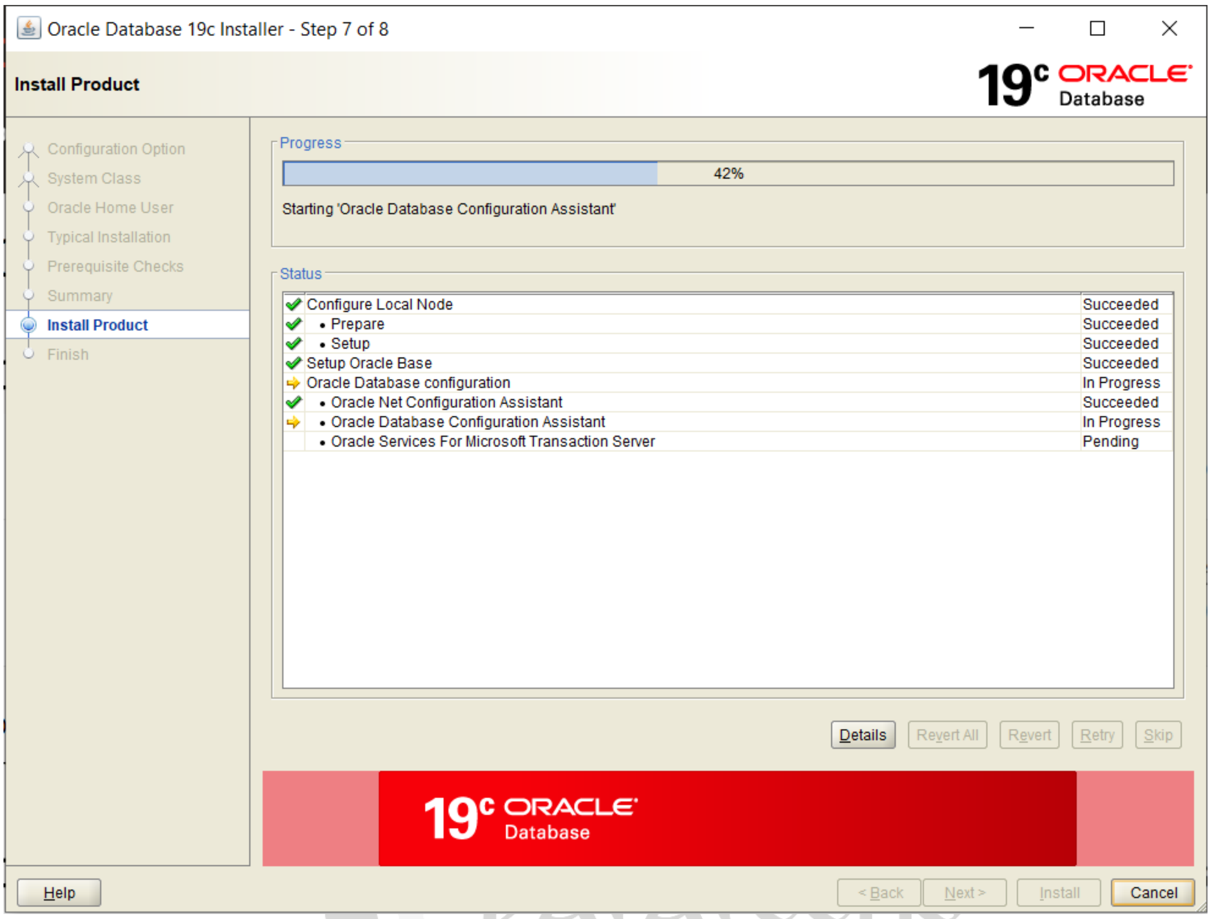
9. In the next screen, a summary will be displayed. Click on 'Install' button.



Summary

```
Oracle Database 19c Installer
  Global settings
    Oracle Home User: Virtual Account [Edit]
    Install method: Desktop installation [Edit]
    Database edition: Enterprise Edition (Create and configure a database) [Edit]
    Oracle base: D:\oracle-19c [Edit]
    Software location: D:\oracle-19c\WINDOWS.X64_193000_db_home
    OraMTS Port Number: 49152
  Inventory information
    Inventory location: C:\Program Files\Oracle\Inventory
  Database information
    Configuration: General Purpose / Transaction Processing
    Global database name: orcl.Dlink [Edit]
    Oracle system identifier (SID): orcl [Edit]
    Pluggable database name: orcl1 [Edit]
    Allocated memory: 6123 MB
    Automatic memory management option: FALSE
    Database character set : AL32UTF8 (Unicode UTF-8 Universal character set) [Edit]
    Management method: Database express
    Database storage mechanism: File system
    Database file location: D:\oracle-19c\oradata [Edit]
    Recovery: Disabled
```

10. The installation will take place a few minutes and hence we have to wait.


Oracle Database 19c Installer - Step 7 of 8

Install Product

Progress

42%

Starting 'Oracle Database Configuration Assistant'

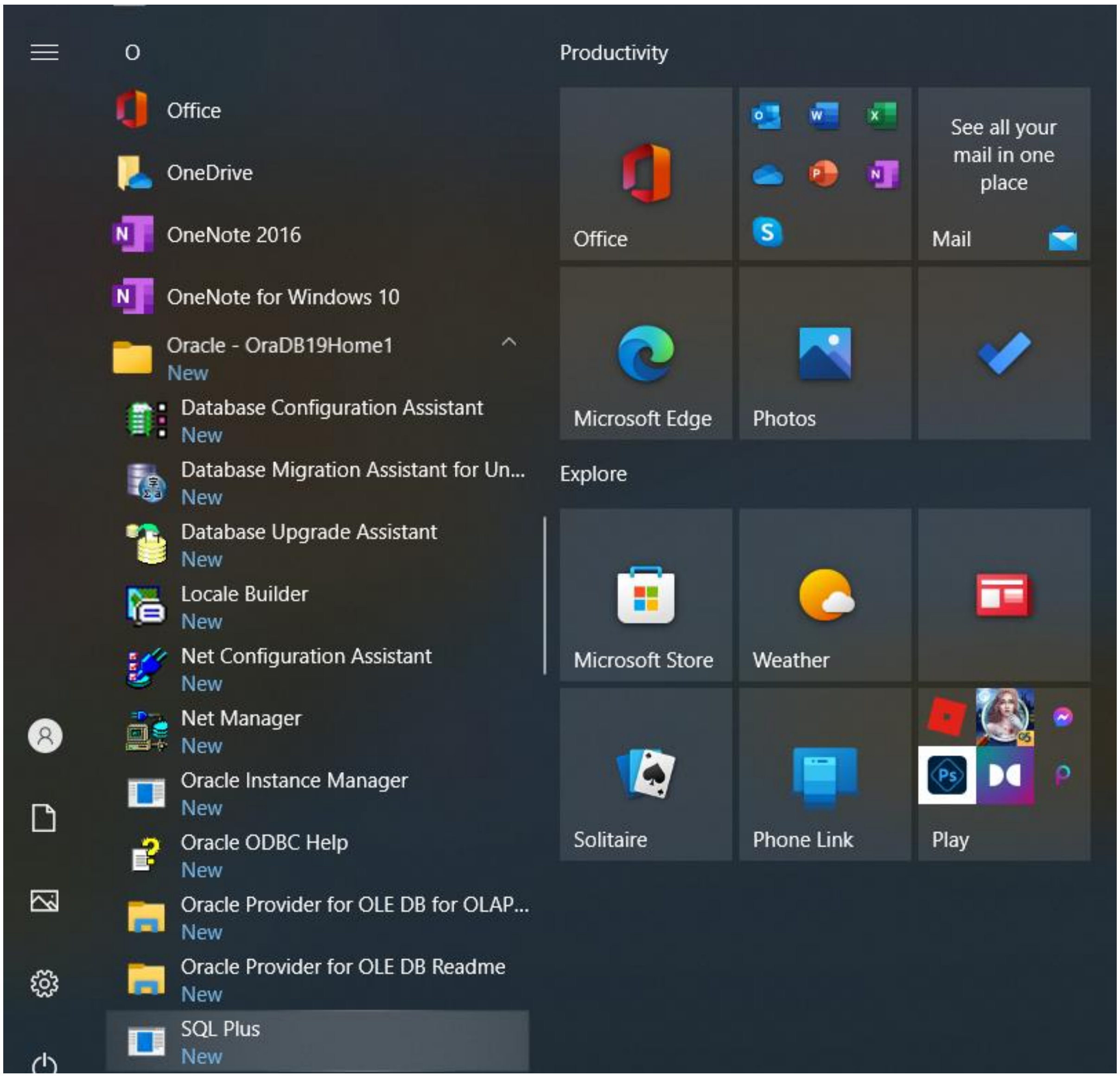
Status

✓ Configure Local Node	Succeeded
✓ • Prepare	Succeeded
✓ • Setup	Succeeded
✓ Setup Oracle Base	Succeeded
⇨ Oracle Database configuration	In Progress
✓ • Oracle Net Configuration Assistant	Succeeded
⇨ • Oracle Database Configuration Assistant	In Progress
• Oracle Services For Microsoft Transaction Server	Pending

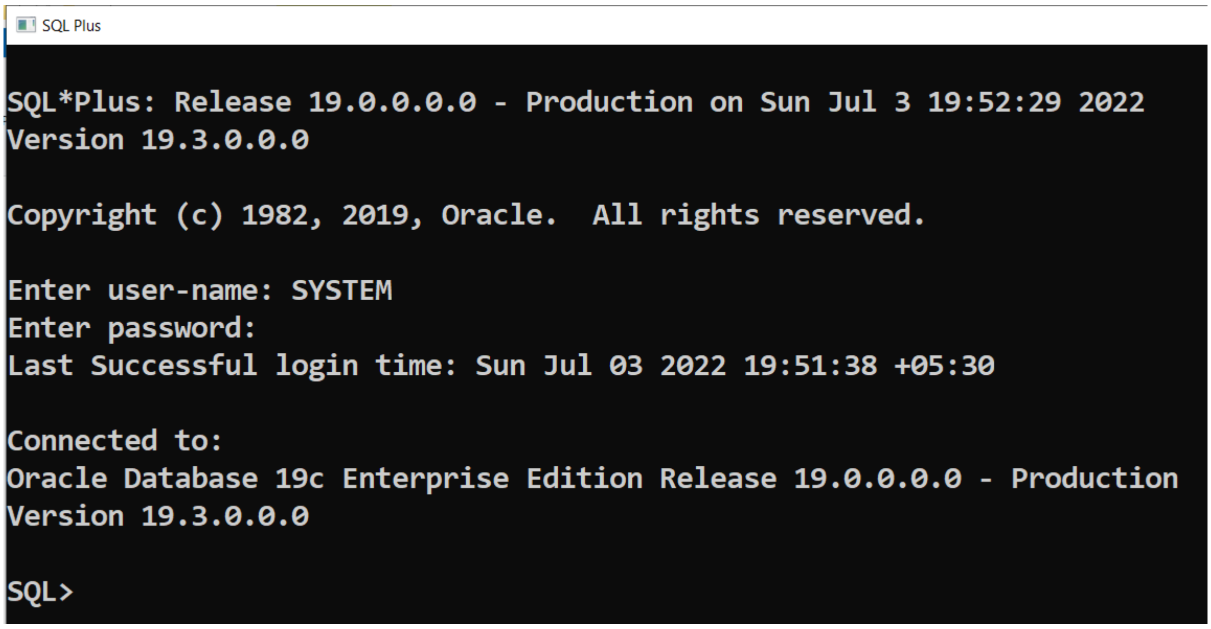
11. Finally, we see a message 'The configuration of Oracle Database was successful'. Click on 'Close' button.

<u>TESTING THE INSTALLATION</u>

1. Once Oracle 19c Installation is complete, you can find a folder by the name 'Oracle – OraDB19Home1' in the Windows Apps. This can be verified by clicking on the Windows 'Start' button and look in the folder names starting with 'O'.

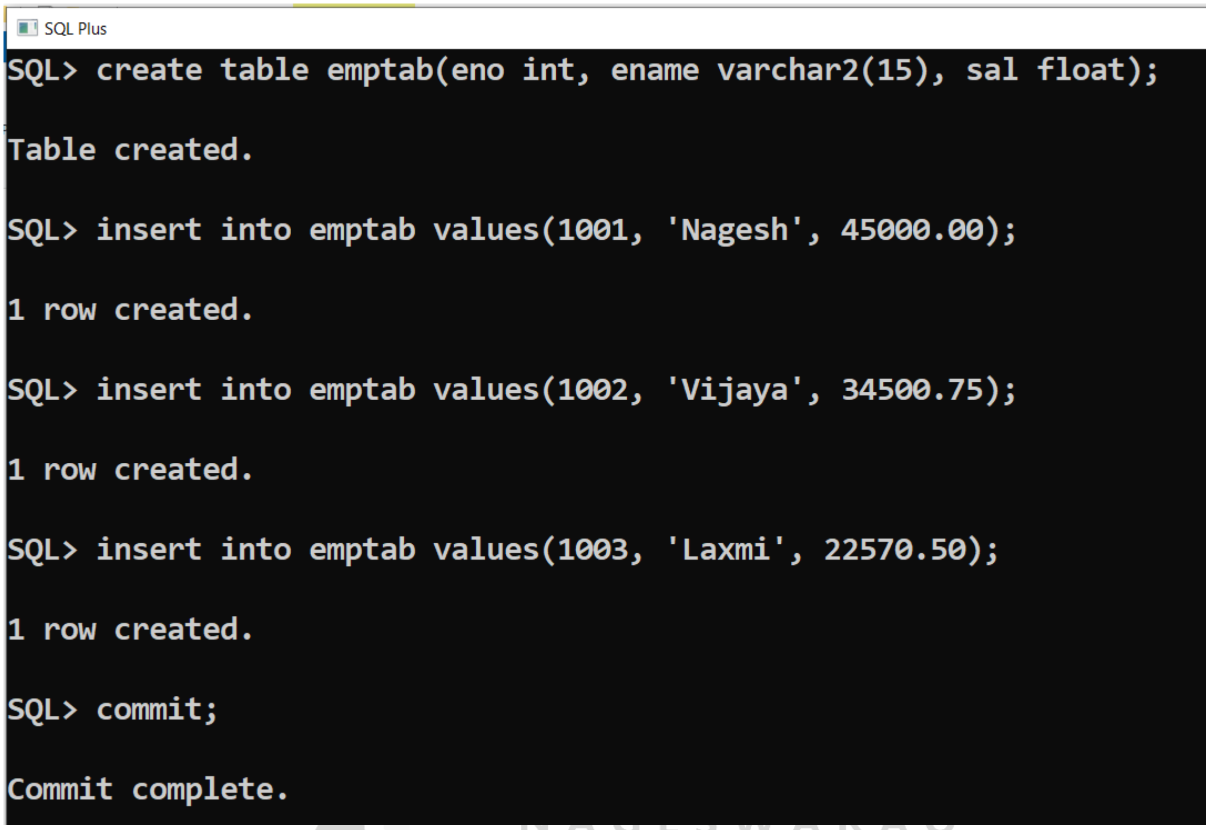


2. Click on 'SQL Plus' and then type User name 'SYSTEM' and Password as entered at the time of Installation.



```
SQL*Plus: Release 19.0.0.0.0 - Production on Sun Jul 3 19:52:29 2022
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Enter user-name: SYSTEM
Enter password:
Last Successful login time: Sun Jul 03 2022 19:51:38 +05:30

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL>
```

3. We can type any SQL commands at the SQL prompt. Try to create a table 'emptab' as shown.



```
SQL> create table emptab(eno int, ename varchar2(15), sal float);

Table created.

SQL> insert into emptab values(1001, 'Nagesh', 45000.00);

1 row created.

SQL> insert into emptab values(1002, 'Vijaya', 34500.75);

1 row created.

SQL> insert into emptab values(1003, 'Laxmi', 22570.50);

1 row created.

SQL> commit;

Commit complete.
```

4. You can view all the rows of emptab by typing:

SQL> select * from emptab;

5. Finally, you can come out of Oracle, by typing:

SQL> exit;

INSTALLATION OF ORACLE CONNECTOR

We have to install cx_Oracle connector that helps to connect a Python program with the Oracle database. For this purpose, we should use 'pip' command, as:

C:\> pip install cx_Oracle

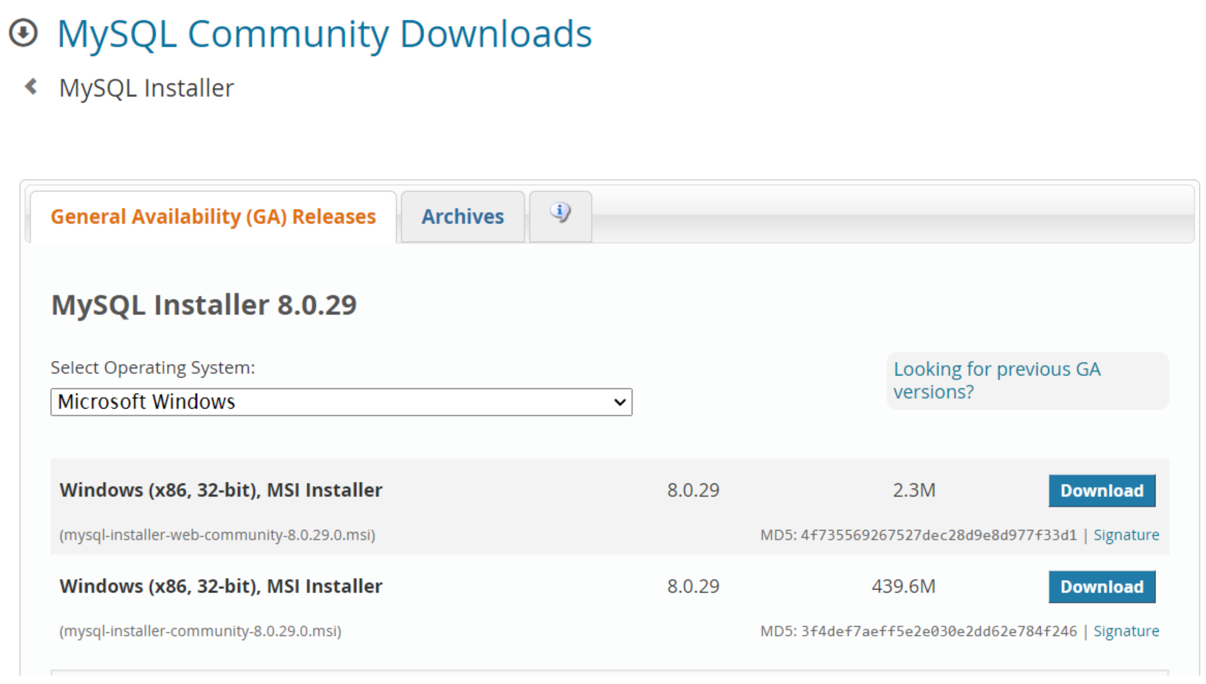
NOTE: cx_Oracle connector version 8.3.0 is recommendable.

MYSQL 8.0 INSTALLATION

1. Go to mysql.com website at the following URL:

https://dev.mysql.com/downloads/windows/installer/8.0.html

click on 'Download' button against 'Windows (x86, 32-bit), MSI Installer.



MySQL Community Downloads

MySQL Installer

General Availability (GA) Releases | Archives

MySQL Installer 8.0.29

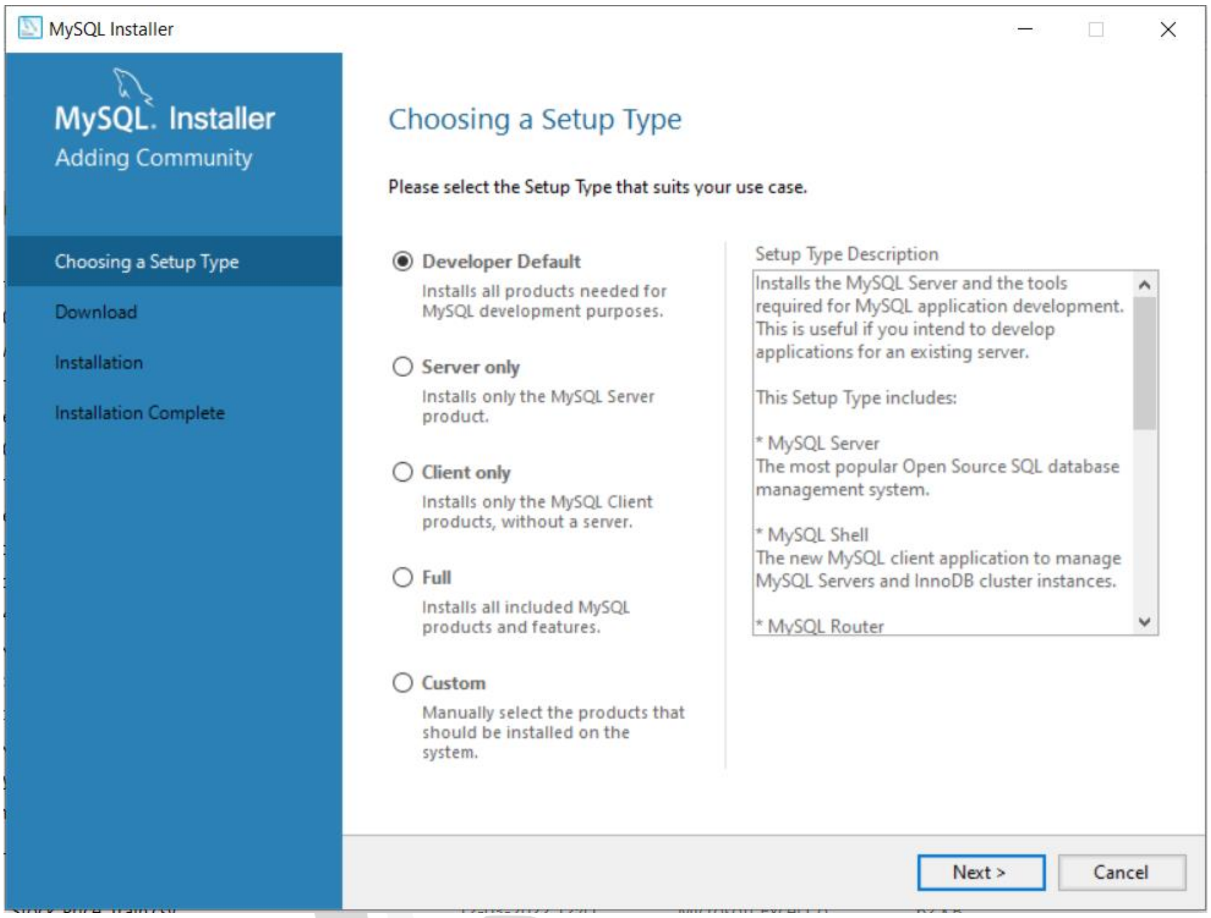
Select Operating System: Microsoft Windows

Looking for previous GA versions?

Windows (x86, 32-bit), MSI Installer	8.0.29	2.3M	Download
(mysql-installer-web-community-8.0.29.0.msi)		MD5: 4f735569267527dec28d9e8d977f33d1 | Signature	
Windows (x86, 32-bit), MSI Installer	8.0.29	439.6M	Download
(mysql-installer-community-8.0.29.0.msi)		MD5: 3f4def7aeff5e2e030e2dd62e784f246 | Signature	

2. In the next page, it will ask for creating an Oracle Web account. You can click on 'No thanks, just start my download' link. The download will start. A file by the name 'mysql-installer-community-8.0.29.0.msi' will download into our system. Double click on this file to run.

3. The MySQL launcher program will launch the Installer and we can see the following screen. Select the radio button with 'Developer Default' and then click on 'Next' button.


MySQL Installer

MySQL. Installer
Adding Community

- Choosing a Setup Type
- Download
- Installation
- Installation Complete

Choosing a Setup Type

Please select the Setup Type that suits your use case.

(●) **Developer Default**
Installs all products needed for MySQL development purposes.

( ) **Server only**
Installs only the MySQL Server product.

( ) **Client only**
Installs only the MySQL Client products, without a server.

( ) **Full**
Installs all included MySQL products and features.

( ) **Custom**
Manually select the products that should be installed on the system.

Setup Type Description

Installs the MySQL Server and the tools required for MySQL application development. This is useful if you intend to develop applications for an existing server.

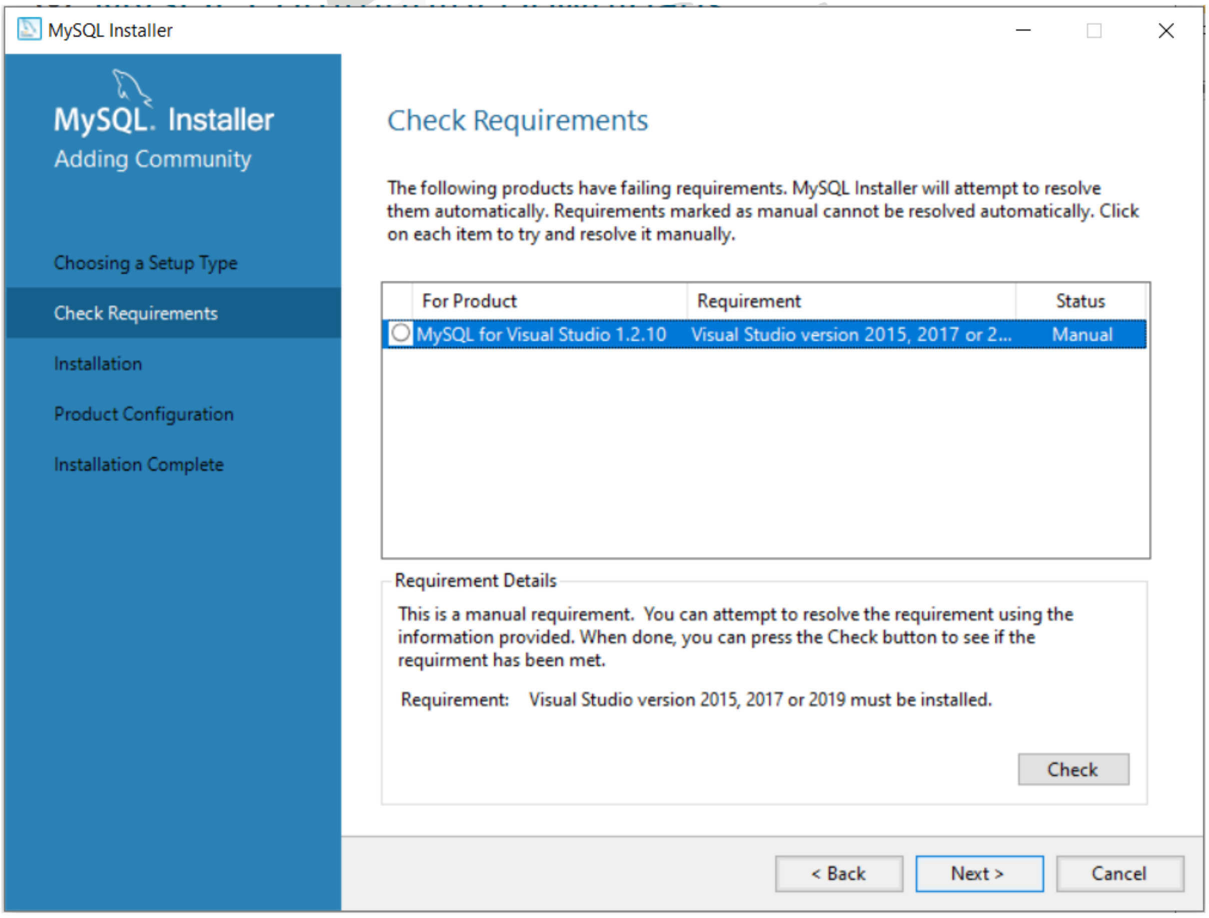
This Setup Type includes:

* MySQL Server
The most popular Open Source SQL database management system.

* MySQL Shell
The new MySQL client application to manage MySQL Servers and InnoDB cluster instances.

* MySQL Router

4. In the next screen, select ‘MySQL for Visual Studio’ radio button and click on ‘Next’ button.


MySQL Installer

MySQL. Installer
Adding Community

- Choosing a Setup Type
- Check Requirements
- Installation
- Product Configuration
- Installation Complete

Check Requirements

The following products have failing requirements. MySQL Installer will attempt to resolve them automatically. Requirements marked as manual cannot be resolved automatically. Click on each item to try and resolve it manually.

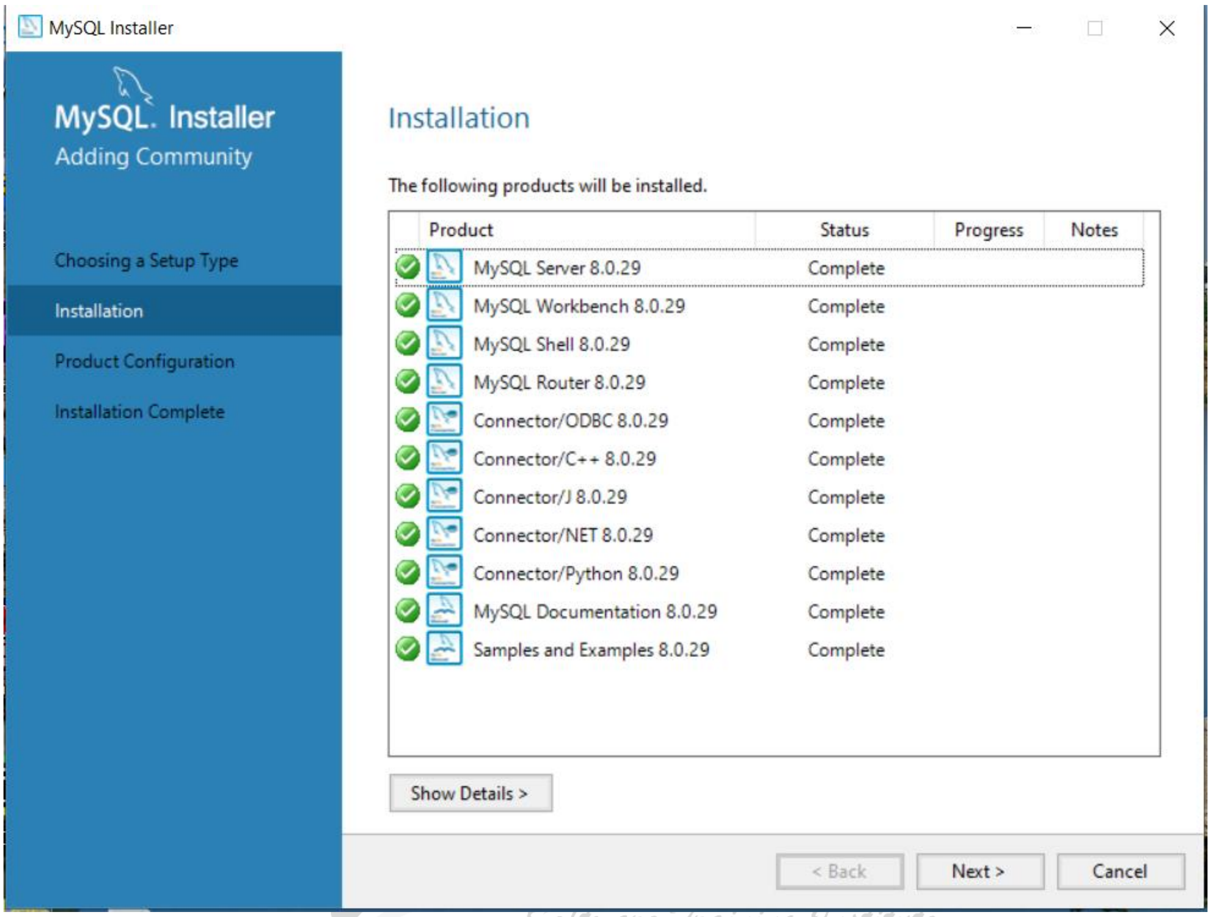
For Product	Requirement	Status
MySQL for Visual Studio 1.2.10	Visual Studio version 2015, 2017 or 2...	Manual

Requirement Details

This is a manual requirement. You can attempt to resolve the requirement using the information provided. When done, you can press the Check button to see if the requirment has been met.

Requirement: Visual Studio version 2015, 2017 or 2019 must be installed.

5. In the next screen, click on ‘Execute’ button to start Installation. It will show percentage of progress of each product being installed. Click on ‘Next’ button once all products installation is complete.

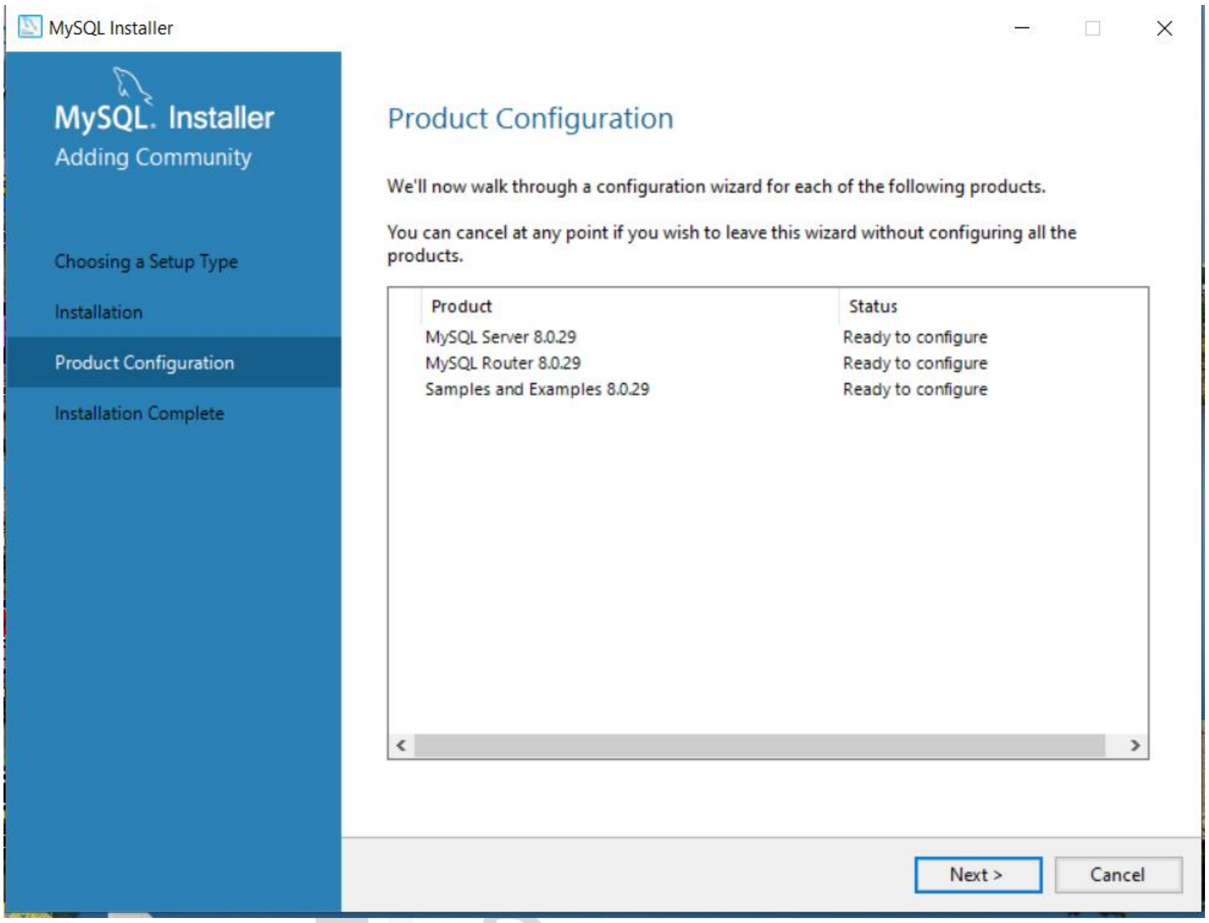


Installation

The following products will be installed.

Product	Status	Progress	Notes
MySQL Server 8.0.29	Complete		
MySQL Workbench 8.0.29	Complete		
MySQL Shell 8.0.29	Complete		
MySQL Router 8.0.29	Complete		
Connector/ODBC 8.0.29	Complete		
Connector/C++ 8.0.29	Complete		
Connector/J 8.0.29	Complete		
Connector/NET 8.0.29	Complete		
Connector/Python 8.0.29	Complete		
MySQL Documentation 8.0.29	Complete		
Samples and Examples 8.0.29	Complete		

6. Click on ‘Next’ button in the next page.


MySQL Installer

MySQL. Installer
Adding Community

Choosing a Setup Type
Installation
Product Configuration
Installation Complete

Product Configuration

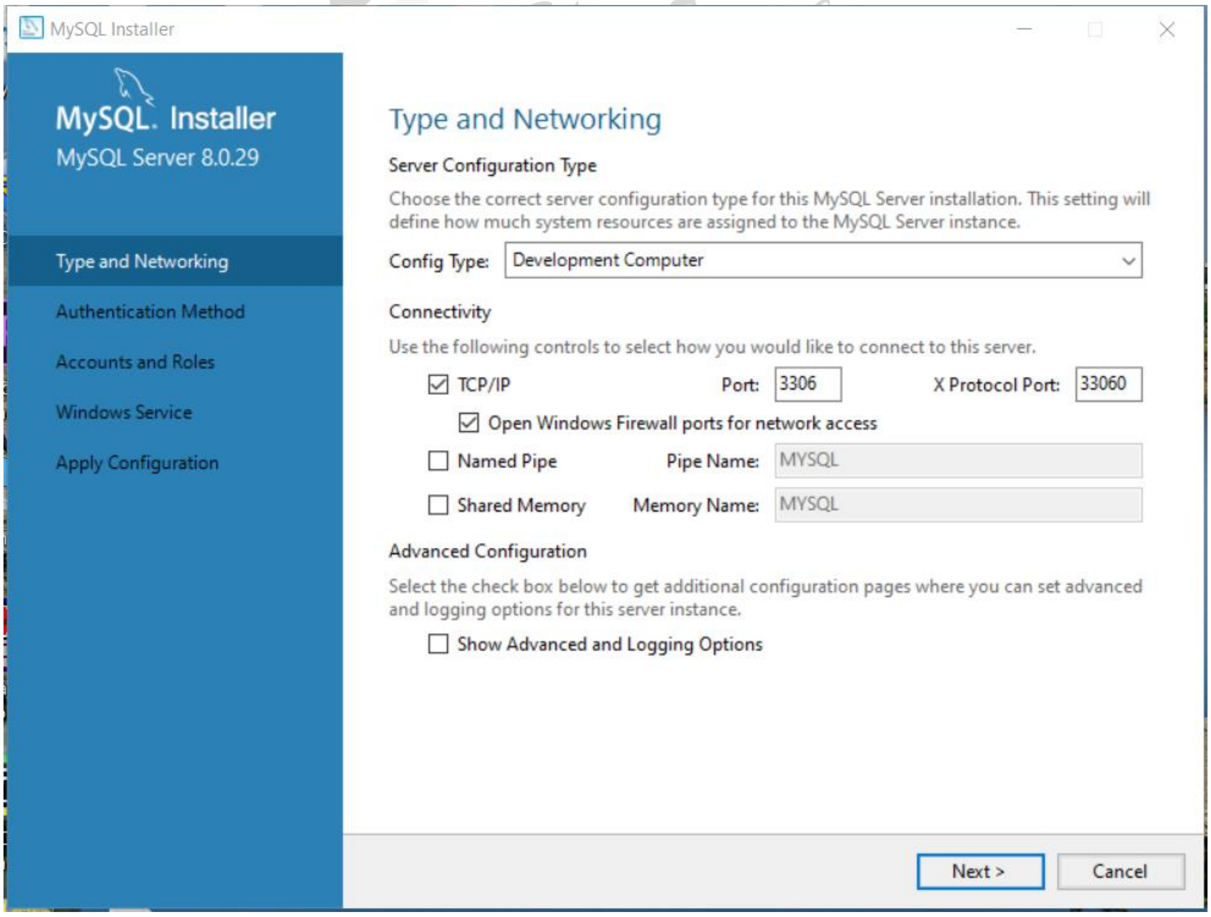
We'll now walk through a configuration wizard for each of the following products.

You can cancel at any point if you wish to leave this wizard without configuring all the products.

Product	Status
MySQL Server 8.0.29	Ready to configure
MySQL Router 8.0.29	Ready to configure
Samples and Examples 8.0.29	Ready to configure

Next > Cancel

7. In the next page, ‘Server Configuration Type’ will be displayed. Click on ‘Next’ button.


MySQL Installer

MySQL. Installer
MySQL Server 8.0.29

Type and Networking
Authentication Method
Accounts and Roles
Windows Service
Apply Configuration

Type and Networking

Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type: Development Computer

Connectivity

Use the following controls to select how you would like to connect to this server.

☑ TCP/IP Port: 3306 X Protocol Port: 33060

☑ Open Windows Firewall ports for network access

☐ Named Pipe Pipe Name: MYSQL

☐ Shared Memory Memory Name: MYSQL

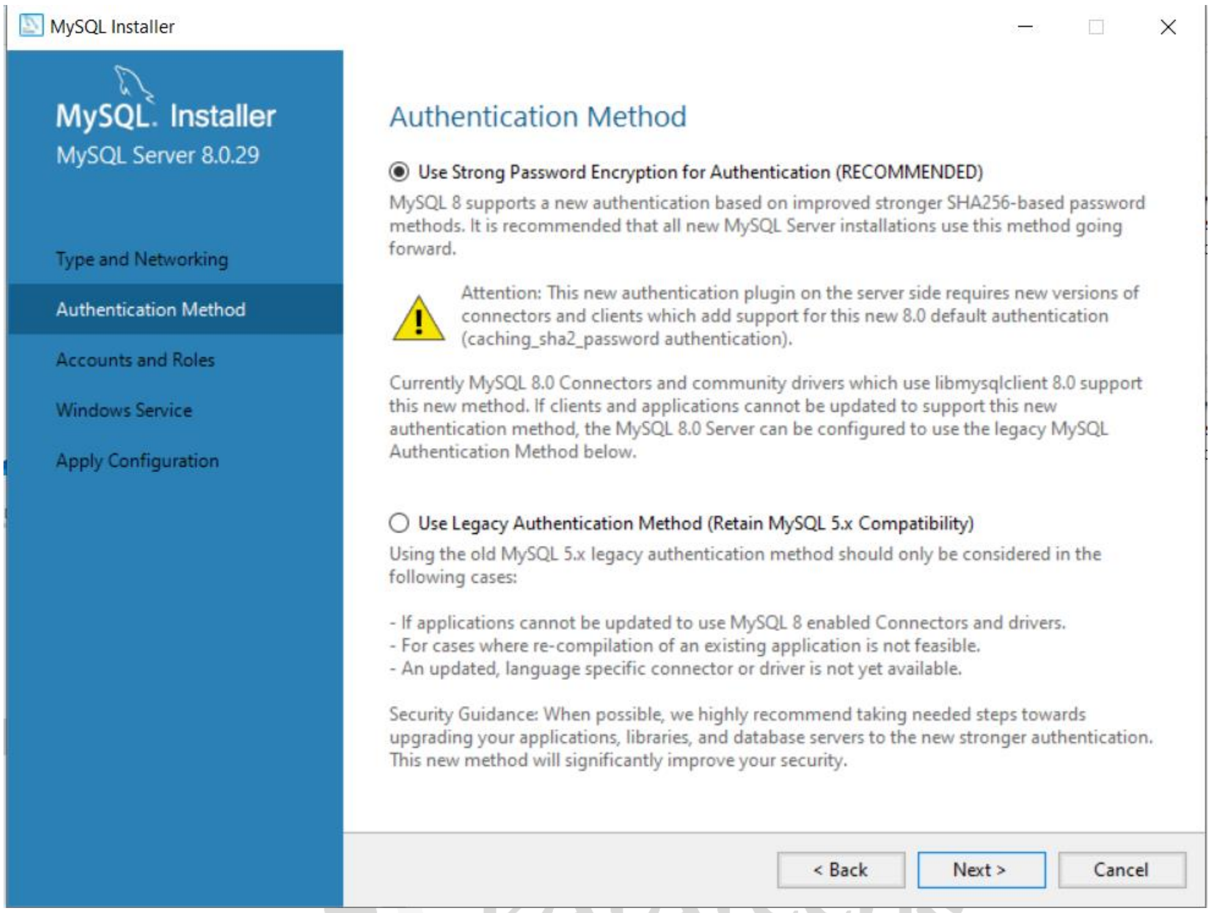
Advanced Configuration

Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.

☐ Show Advanced and Logging Options

Next > Cancel

8. In the ‘Authentication Method’ page, click on ‘Next’ button.



Authentication Method

● Use Strong Password Encryption for Authentication (RECOMMENDED)

MySQL 8 supports a new authentication based on improved stronger SHA256-based password methods. It is recommended that all new MySQL Server installations use this method going forward.

Attention: This new authentication plugin on the server side requires new versions of connectors and clients which add support for this new 8.0 default authentication (caching_sha2_password authentication).

Currently MySQL 8.0 Connectors and community drivers which use libmysqlclient 8.0 support this new method. If clients and applications cannot be updated to support this new authentication method, the MySQL 8.0 Server can be configured to use the legacy MySQL Authentication Method below.

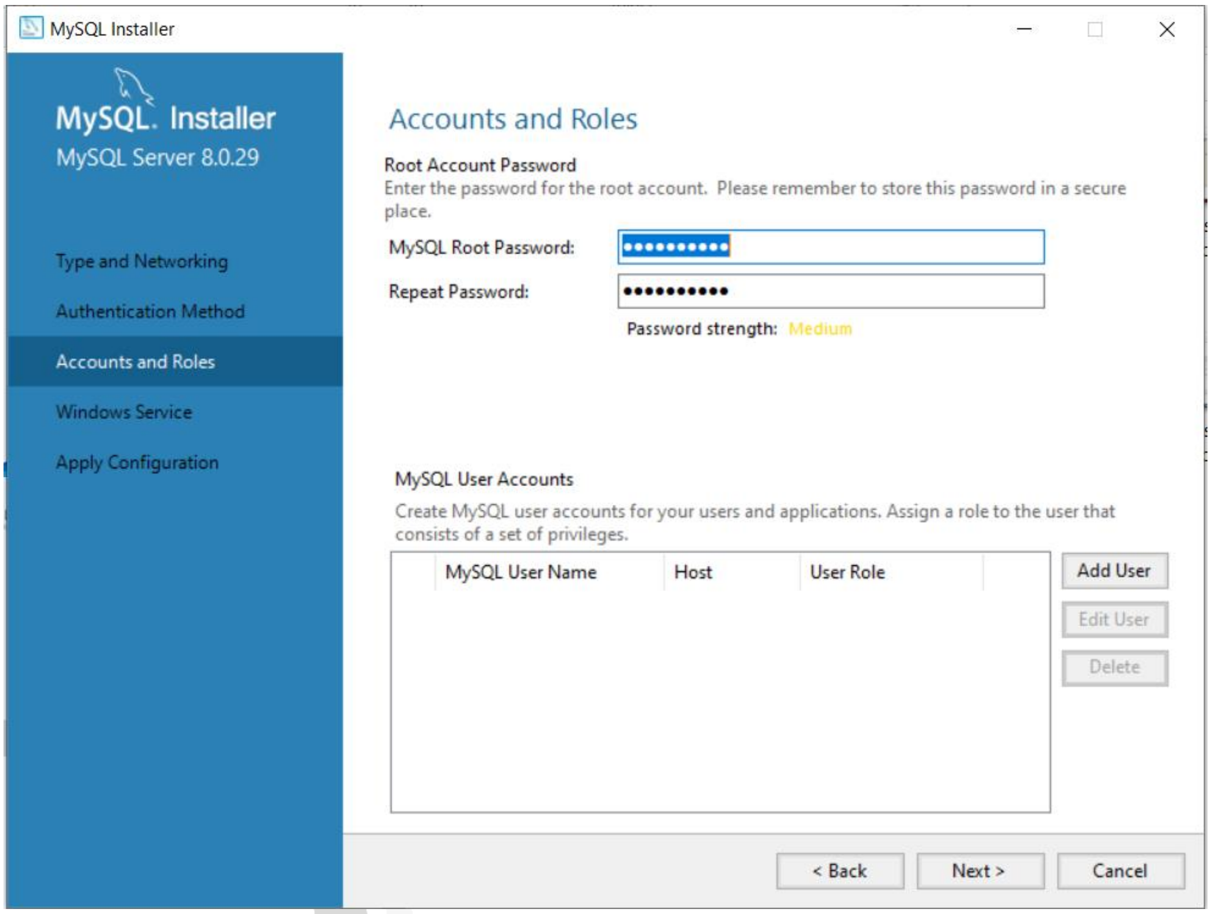
○ Use Legacy Authentication Method (Retain MySQL 5.x Compatibility)

Using the old MySQL 5.x legacy authentication method should only be considered in the following cases:

- If applications cannot be updated to use MySQL 8 enabled Connectors and drivers.
- For cases where re-compilation of an existing application is not feasible.
- An updated, language specific connector or driver is not yet available.

Security Guidance: When possible, we highly recommend taking needed steps towards upgrading your applications, libraries, and database servers to the new stronger authentication. This new method will significantly improve your security.

9. In the next page, enter MySQL Password. You must remember this password so that you can use this in future.


MySQL Installer

MySQL Installer
MySQL Server 8.0.29

- Type and Networking
- Authentication Method
- Accounts and Roles
- Windows Service
- Apply Configuration

Accounts and Roles

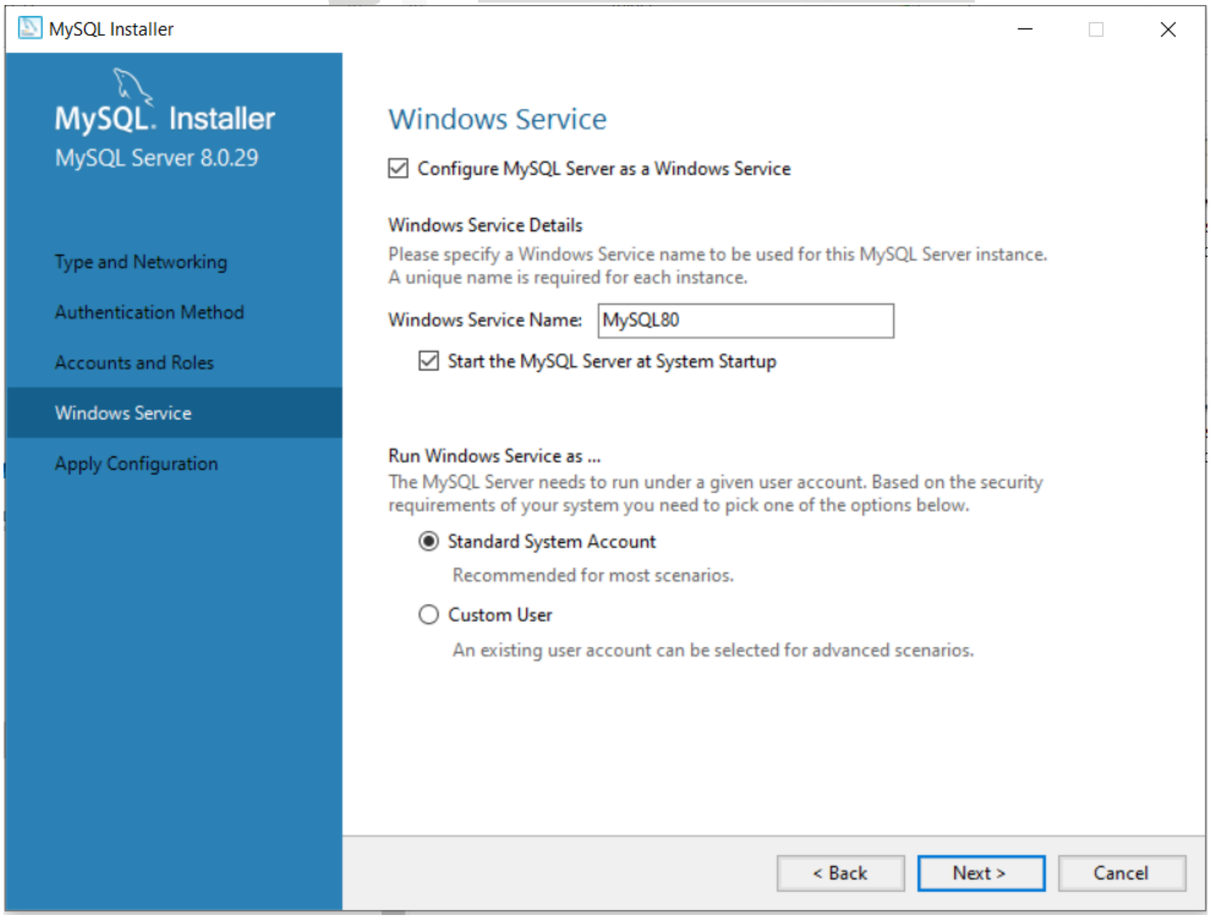
Root Account Password
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password: ••••••••••
Repeat Password: ••••••••••
Password strength: Medium

MySQL User Accounts
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL User Name	Host	User Role

10. In the next page, click on ‘Next’ button.


MySQL Installer

MySQL Installer
MySQL Server 8.0.29

- Type and Networking
- Authentication Method
- Accounts and Roles
- Windows Service
- Apply Configuration

Windows Service

☑ Configure MySQL Server as a Windows Service

Windows Service Details
Please specify a Windows Service name to be used for this MySQL Server instance.
A unique name is required for each instance.

Windows Service Name: MySQL80

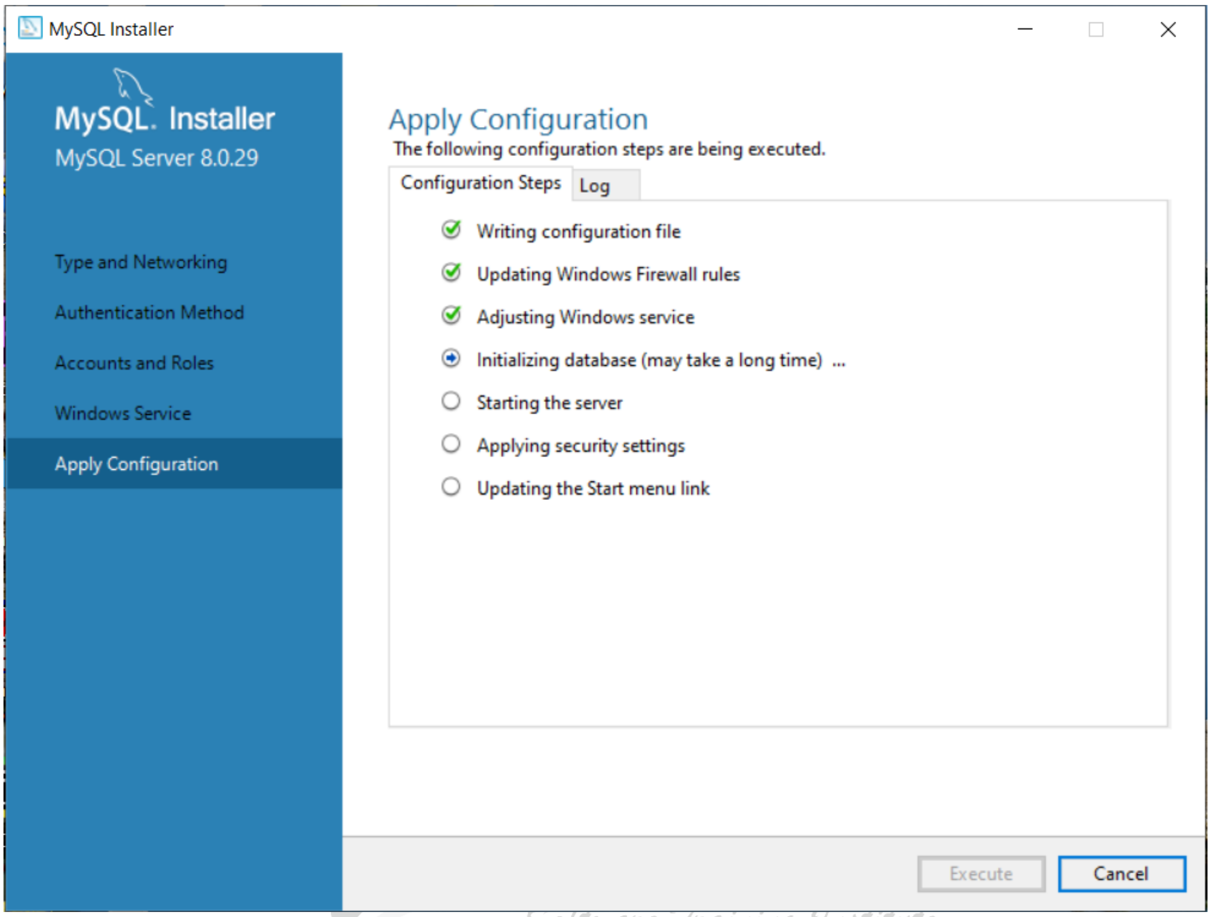
☑ Start the MySQL Server at System Startup

Run Windows Service as ...
The MySQL Server needs to run under a given user account. Based on the security requirements of your system you need to pick one of the options below.

[✓ Standard System Account]
Recommended for most scenarios.

Custom User
An existing user account can be selected for advanced scenarios.

11. In the next page, click on ‘Execute’ button. The configurations are applied now. When you see all ‘tick’ marks, click on ‘Finish’ button.


MySQL Installer

MySQL. Installer
MySQL Server 8.0.29

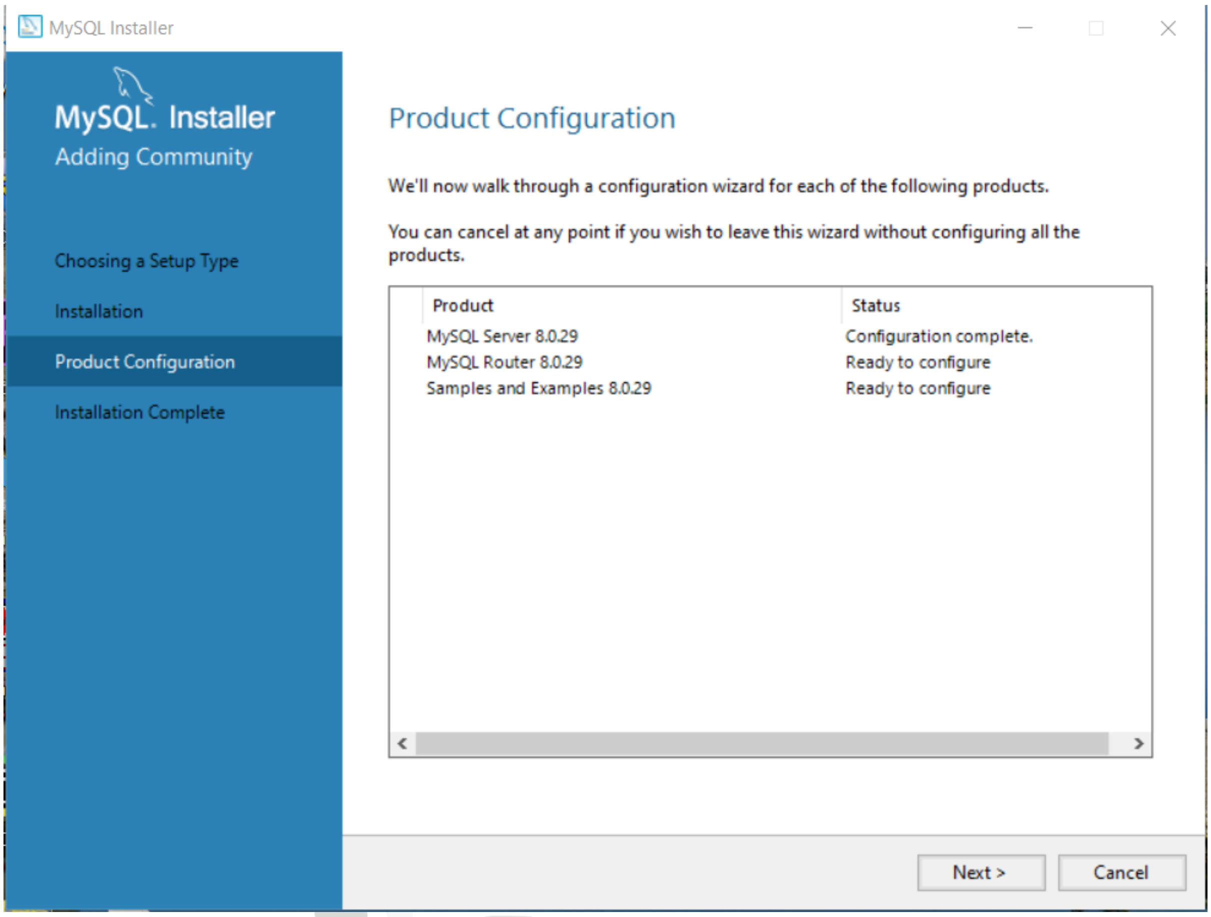
Type and Networking
Authentication Method
Accounts and Roles
Windows Service
Apply Configuration

Apply Configuration
The following configuration steps are being executed.

Configuration Steps | Log

- ✓ Writing configuration file
- ✓ Updating Windows Firewall rules
- ✓ Adjusting Windows service
- ● Initializing database (may take a long time) ...
- ○ Starting the server
- ○ Applying security settings
- ○ Updating the Start menu link

12. The next page with the heading ‘Product Configuration’ will appear. Click on ‘Next’ button.



Product Configuration

We'll now walk through a configuration wizard for each of the following products.

You can cancel at any point if you wish to leave this wizard without configuring all the products.

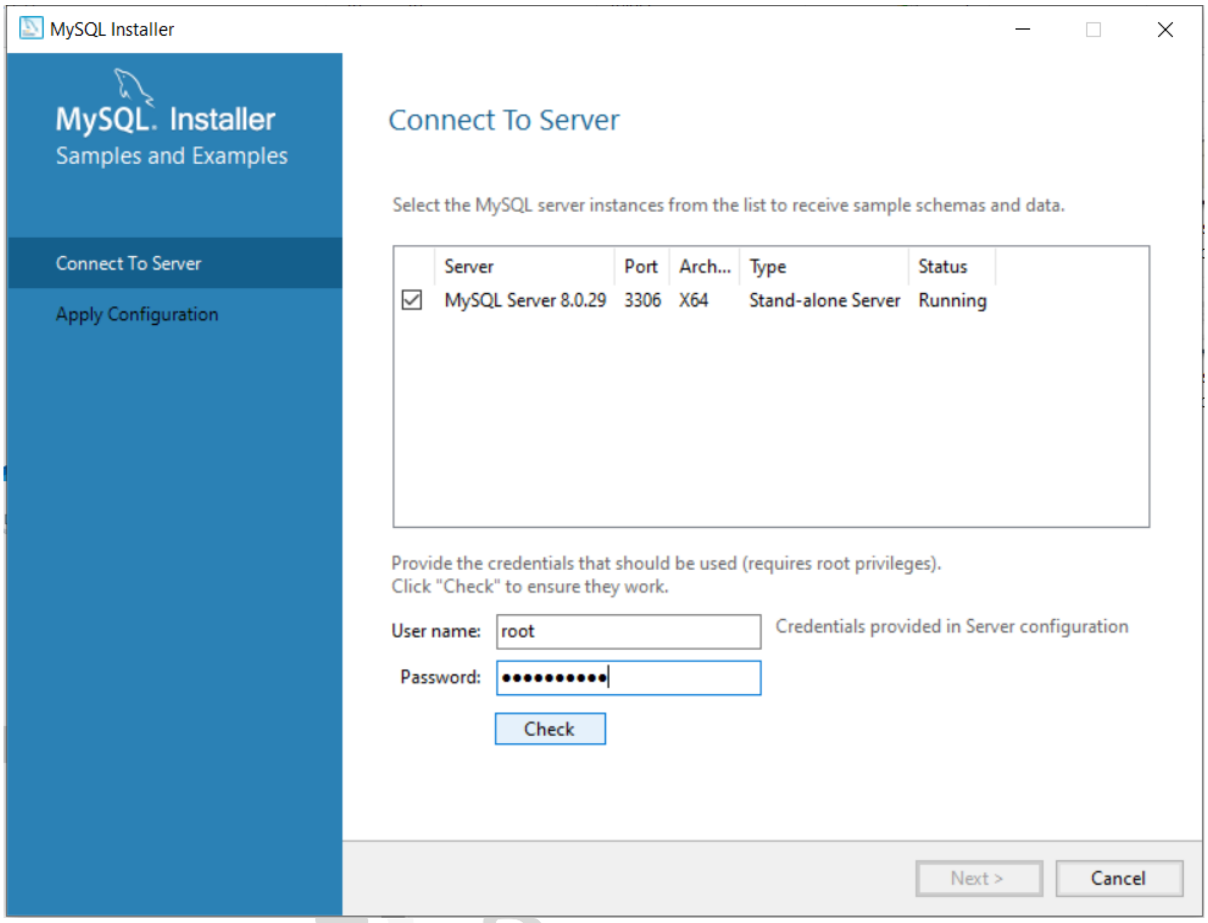
Product	Status
MySQL Server 8.0.29	Configuration complete.
MySQL Router 8.0.29	Ready to configure
Samples and Examples 8.0.29	Ready to configure

13. In the next page, click on 'Finish' button. Also, in the next page, click on 'Next' button. It will display:

User name: root

Password: rnr_Mysql1 (Enter this)

Type the password as entered previously. Then click on 'Check' button. Then a green message 'connection succeeded' will appear. After that, click on 'Next' button.



Connect To Server

Select the MySQL server instances from the list to receive sample schemas and data.

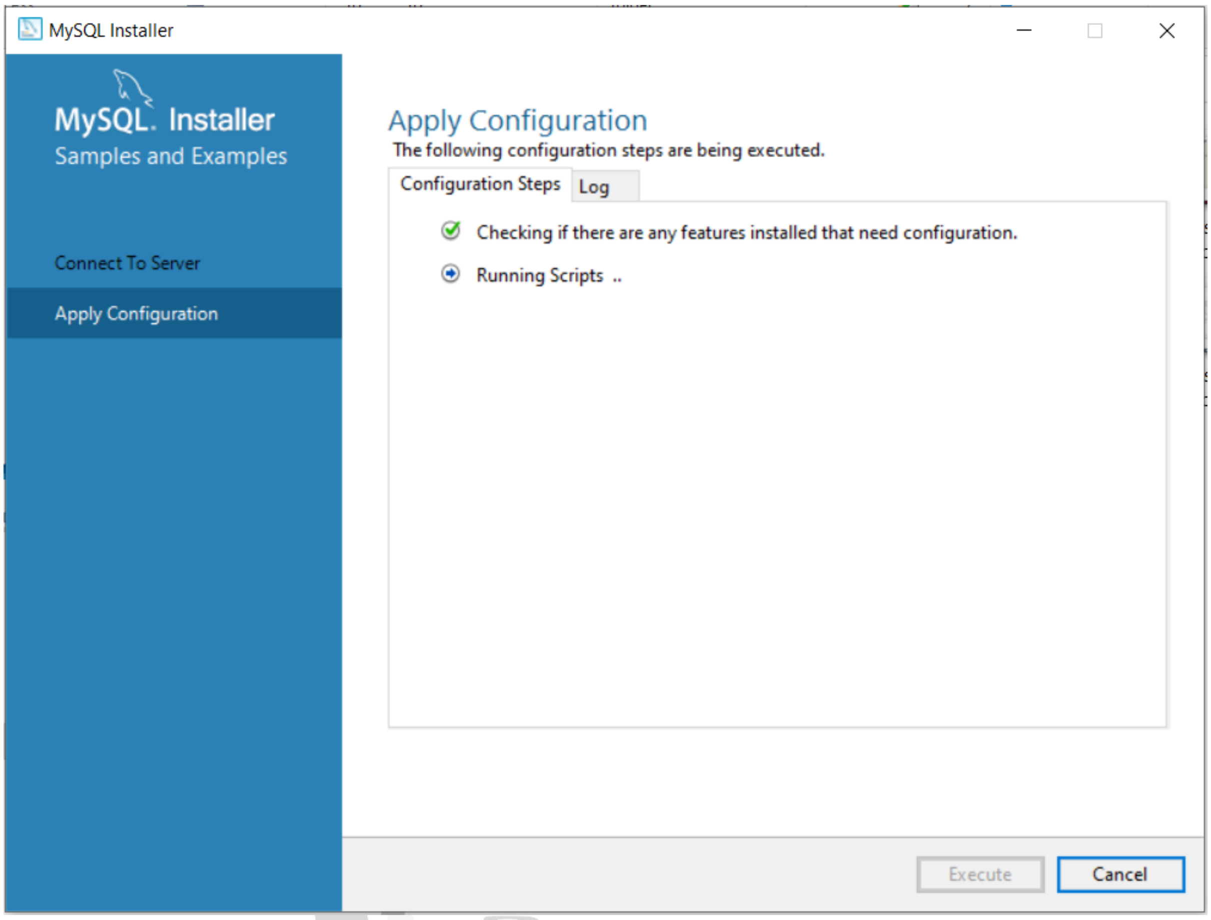
	Server	Port	Arch...	Type	Status
☑	MySQL Server 8.0.29	3306	X64	Stand-alone Server	Running

Provide the credentials that should be used (requires root privileges).
Click "Check" to ensure they work.

User name: root Credentials provided in Server configuration

Password: ●●●●●●●●●●●

14. In the next page, click on ‘Execute’ button. When the tick marks appear, click on ‘Finish’ button.


MySQL Installer

MySQL. Installer
Samples and Examples

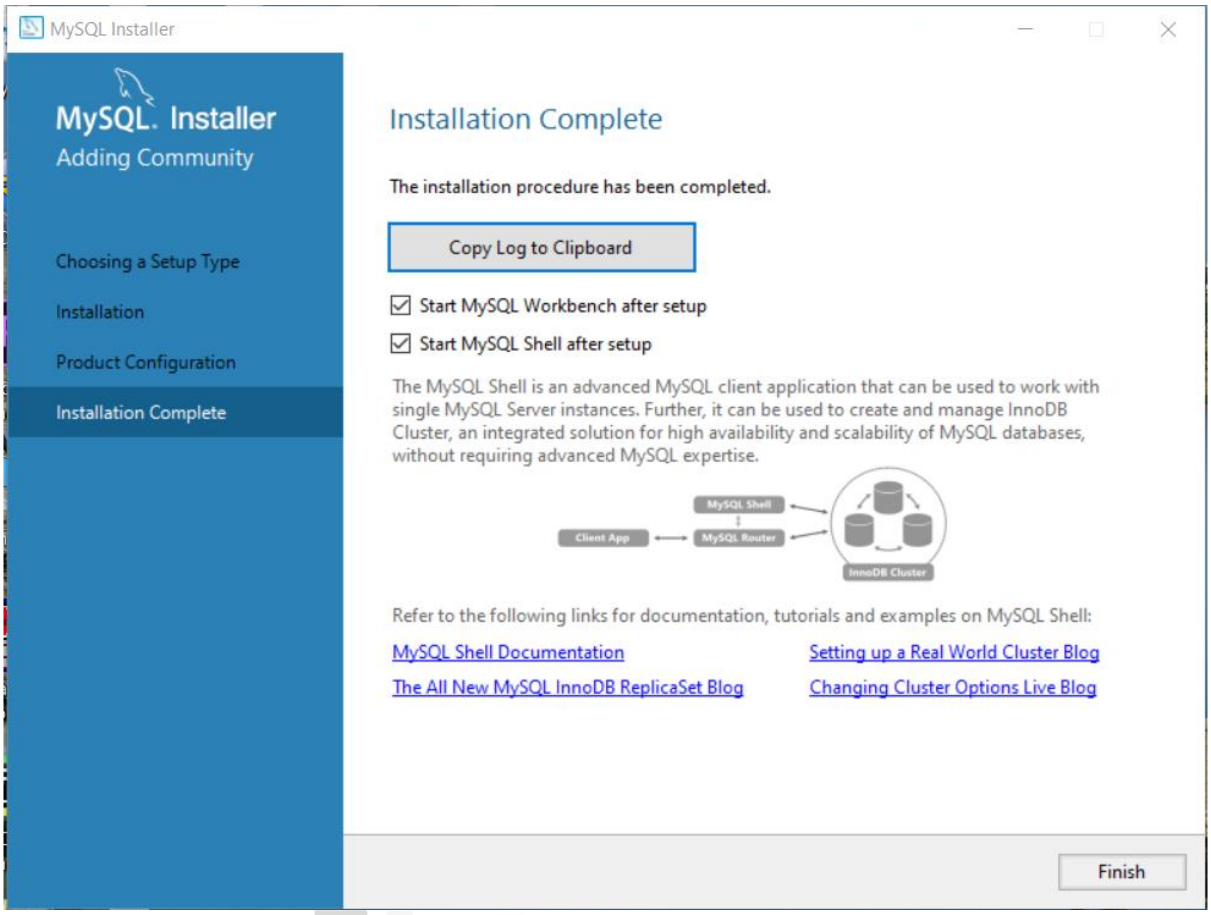
Connect To Server
Apply Configuration

Apply Configuration
The following configuration steps are being executed.

Configuration Steps | Log

✓ Checking if there are any features installed that need configuration.
● Running Scripts ..

15. In the next page, click on ‘Next’ button. In the next page, click on ‘Finish’ button. This will complete the Installation of MySQL. Close any windows that may appear later.



Installation Complete

The installation procedure has been completed.

☑ Start MySQL Workbench after setup

☑ Start MySQL Shell after setup

The MySQL Shell is an advanced MySQL client application that can be used to work with single MySQL Server instances. Further, it can be used to create and manage InnoDB Cluster, an integrated solution for high availability and scalability of MySQL databases, without requiring advanced MySQL expertise.

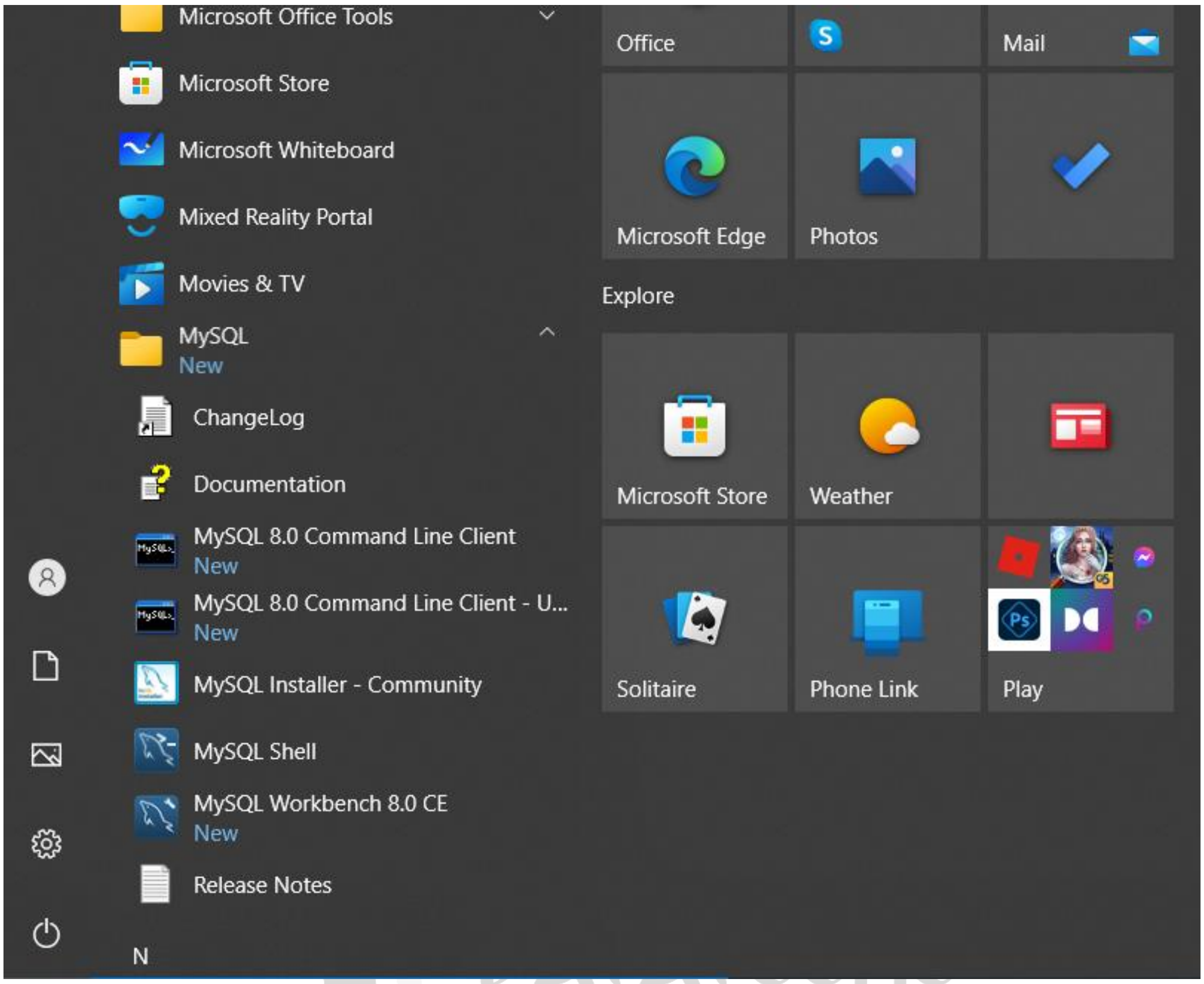
Refer to the following links for documentation, tutorials and examples on MySQL Shell:

MySQL Shell Documentation Setting up a Real World Cluster Blog

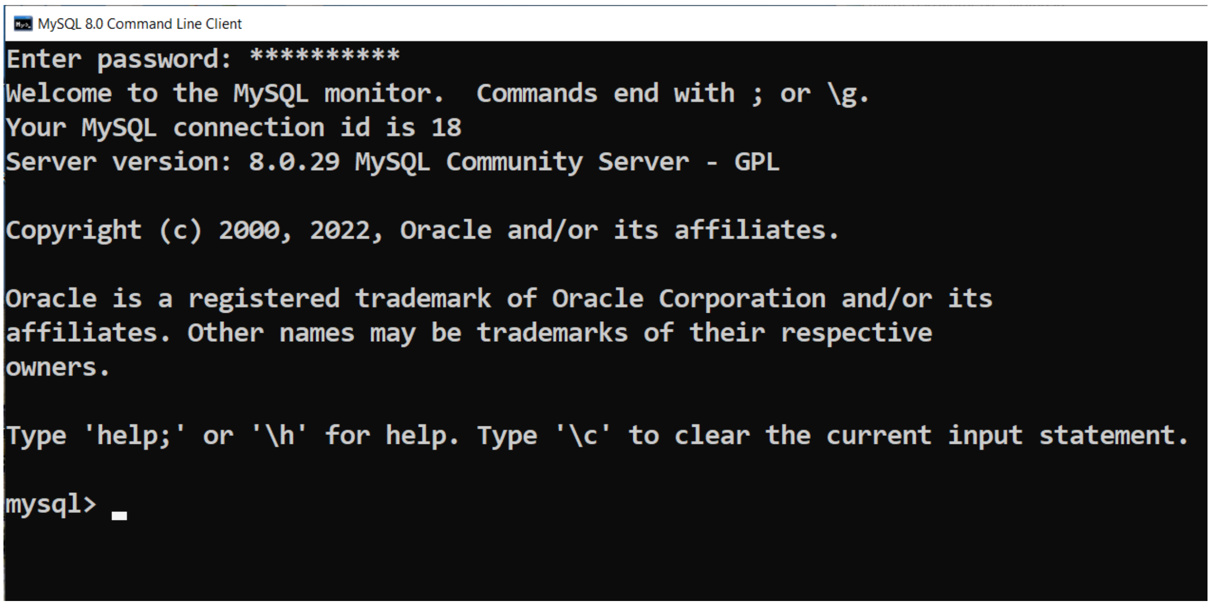
The All New MySQL InnoDB ReplicaSet Blog Changing Cluster Options Live Blog

CHECKING THE INSTALLATION

1. When the installation is successful, we can see a folder by the name ‘MySQL’ in Windows. Click on ‘Start’ button in Windows and then click on this folder.

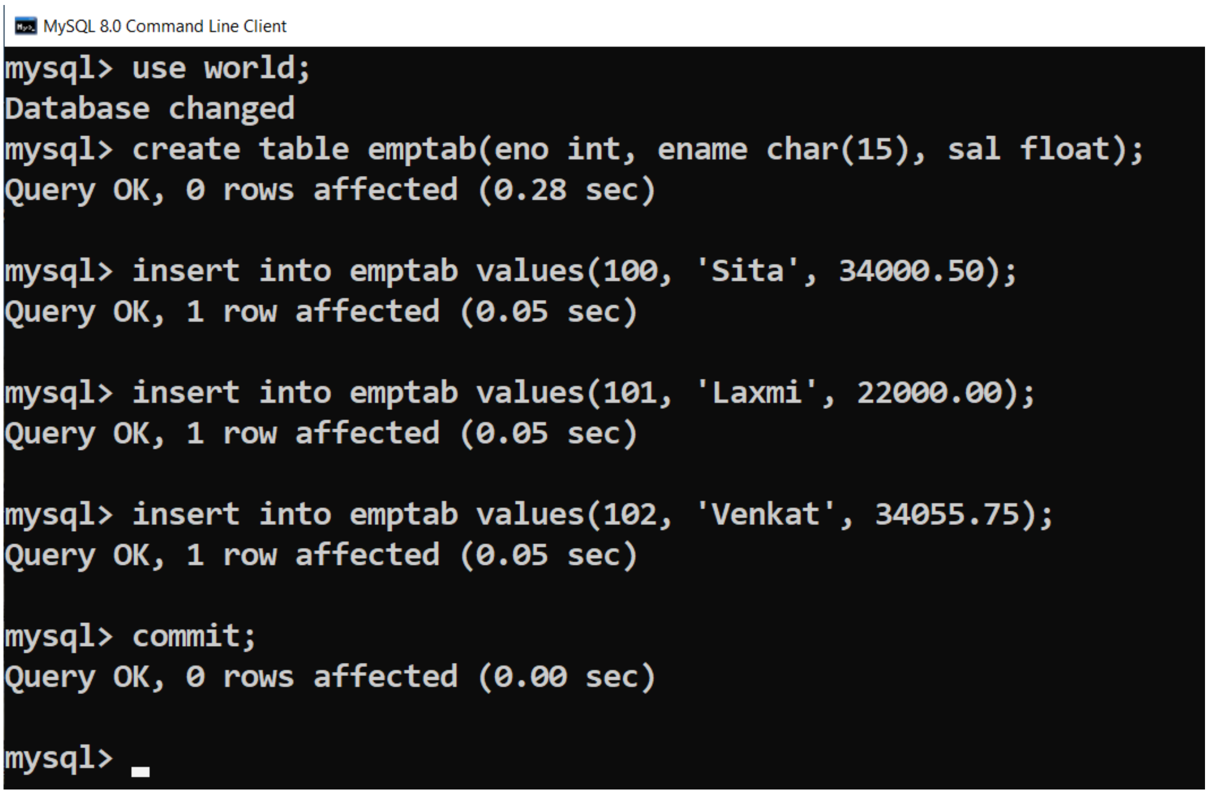


2. Inside 'MySQL' folder, we can see the option 'MySQL 8.0 Command Line Client'. Click on it. It will ask for password. Enter the password which you have given at the time of installation. You can see 'mysql' prompt.



```
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18
Server version: 8.0.29 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

3. Now we can enter any MySQL commands. To quit from MySQL, we can type exit; at the prompt.


MySQL 8.0 Command Line Client

```
mysql> use world;
Database changed
mysql> create table emptab(eno int, ename char(15), sal float);
Query OK, 0 rows affected (0.28 sec)

mysql> insert into emptab values(100, 'Sita', 34000.50);
Query OK, 1 row affected (0.05 sec)

mysql> insert into emptab values(101, 'Laxmi', 22000.00);
Query OK, 1 row affected (0.05 sec)

mysql> insert into emptab values(102, 'Venkat', 34055.75);
Query OK, 1 row affected (0.05 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> _
```

INSTALLATION OF CONNECTOR

To connect to MySQL database, we need pymysql connector. This can be installed using pip command, as:

```
C:\> pip install pymysql
```

NOTE: We are using pymysql version 1.0.2.

FUNDAMENTALS OF PYTHON

Python was developed by Guido Van Rossum in the year 1991.

Python is a high level programming language that contains features of functional programming language like C and object oriented programming language like Java.

FEATURES OF PYTHON

- **Simple**

Python is a simple programming language because it uses English like sentences in its programs.

- **Easy to learn**

Python uses very few keywords. Its programs use very simple structure.

- **Open source**

Python can be freely downloaded from www.python.org website. Its source code can be read, modified and can be used in programs as desired by the programmers.

- **High level language**

High level languages use English words to develop programs. These are easy to learn and use. Like COBOL, PHP or Java, Python also uses English words in its programs and hence it is called high level programming language.

- **Dynamically typed**

In Python, we need not declare the variables. Depending on the value stored in the variable, Python interpreter internally assumes the datatype.

- **Platform independent**

Hence, Python programs are not dependant on any computer with any operating system. We can use Python on Unix, Linux, Windows, Macintosh, Solaris, OS/2, Amiga, AROS, AS/400, etc. almost all operating systems. This will make Python an ideal programming language for any network or Internet.

- **Portable**

When a program yields same result on any computer in the world, then it is called a portable program. Python programs will give same result since they are platform independent.

- **Procedure and Object oriented**

Python is a procedure oriented as well as object oriented programming language. In procedure oriented programming languages (e.g. C and Pascal), the programs are built using functions and procedures. But in object oriented languages (e.g. C++ and Java), the programs use classes and objects.

An object is anything that exists physically in the real world. An object contains behavior. This behavior is represented by its properties (or attributes) and actions. Properties are represented by variables and actions are performed by methods. So, an object contains variables and methods.

A class represents common behavior of a group of objects. It also contains variables and methods. But a class does not exist physically.

A class can be imagined as a model for creating objects. An object is an instance (physical form) of a class.

- **Interpreted**

First, Python compiler translates the Python program into an intermediate code called *byte code*. This byte code is then executed by PVM. Inside the PVM, an interpreter converts the byte code instructions into machine code so that the processor will understand and run that machine code.

- **Extensible**

There are other flavors of Python where programs from other languages can be integrated into Python. For example, Jython is useful to integrate Java code into Python programs and run on JVM (Java Virtual Machine). Similarly IronPython is useful to integrate .NET programs and libraries into Python programs and run on CLR (Common Language Runtime).

- **Embeddable**

Several applications are already developed in Python which can be integrated into other programming languages like C, C++, Delphi, PHP, Java and .NET. It means programmers can use these applications for their advantage in various software projects.

- **Huge library**

Python has a big library that contains modules which can be used on any Operating system.

- **Scripting language**

A scripting language uses an interpreter to translate the source code into machine code on the fly (while running). Generally, scripting languages perform supporting tasks for a bigger application or software. Python is considered as a scripting language as it is interpreted and it is used on Internet to support other softwares.

- **Database connectivity**

A database represents software that stores and manipulates data. Python provides interfaces to connect its programs to all major databases like Oracle, Sybase, SQL Server or MySql.

- **Scalable**

A program would be scalable if it could be moved to another Operating system or hardware and take full advantage of the new environment in terms of performance.

- **Batteries included**

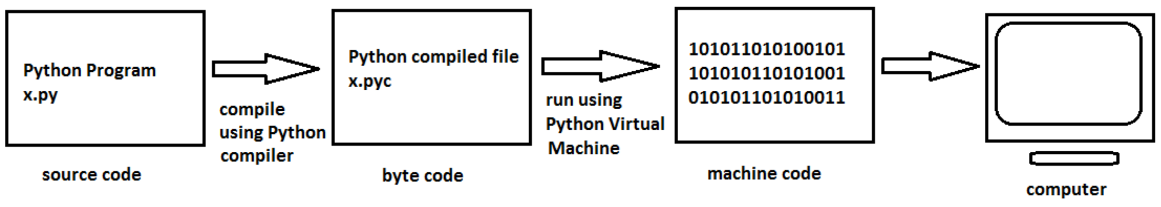
The huge library of Python contains several small applications (or small packages) which are already developed and immediately available to programmers. These libraries are called 'batteries included'. Some interesting batteries or packages are given here:

argparse is a package that represents command-line parsing library.
boto is Amazon web services library.
CherryPy is a Object-oriented HTTP framework.
cryptography offers cryptographic techniques for the programmers
Fiona reads and writes big data files
jellyfish is a library for doing approximate and phonetic matching of strings.
matplotlib is a library for electronics and electrical drawings.
mysql-connector-python is a driver written in Python to connect to MySQL database.
numpy is a package for processing arrays of single or multidimensional type.
pandas is a package for powerful data structures for data analysis, time series and statistics.
Pillow is a Python imaging library.
pyquery represents jquery-like library for Python.
scipy is the scientific library to do scientific and engineering calculations.
Sphinx is the Python documentation generator.
sympy is a package for Computer algebra system (CAS) in Python.
w3lib is a library of web related functions.
whoosh contains fast and pure Python full text indexing, search and spell checking library.

To know the entire list of packages included in Python, one can visit:
https://www.pythonanywhere.com/batteries_included/

PVM

A Python program contains source code (first.py) that is first compiled by Python compiler to produce byte code (first.pyc). This byte code is given to Python Virtual Machine (PVM) which converts the byte code to machine code. This machine code is run by the processor and finally the results are produced.



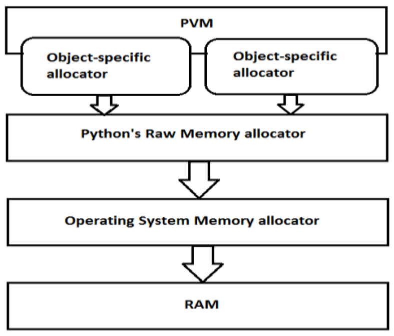
Python Virtual Machine (PVM) is a software that contains an interpreter that converts the byte code into machine code.

PVM is most often called Python interpreter. The PVM of PyPy contains a compiler in addition to the interpreter. This compiler is called Just In Time (JIT) compiler which is useful to speed up execution of the Python program.

Memory management by PVM

Memory allocation and deallocation are done by PVM during runtime. Entire memory is allocated on heap.

We know that the actual memory (RAM) for any program is allocated by the underlying Operating system. On the top of the Operating system, a raw memory allocator oversees whether enough memory is available to it for storing the objects (ex: integers, strings, functions, lists, modules etc). On the top of the raw memory allocator, there are several object-specific allocators operate on the same heap. These memory allocators will implement different types of memory management policies depending on the type of the objects. For example, an integer number should be stored in memory in one way and a string should be stored in a different way. Similarly, when we deal with tuples and dictionaries, they should be stored differently. These issues are taken care by object-specific memory allocators.


PVM
Object-specific allocator
Object-specific allocator
Python's Raw Memory allocator
Operating System Memory allocator
RAM

Garbage collection

A module represents Python code that performs specific task. Garbage collector is a module in Python that is useful to delete objects from memory which are not used in the program. The module that represents the garbage collector is named as *gc*. Garbage collector in the simplest way maintains a count for each object regarding how many times that object is referenced (or used). When an object is referenced twice, its reference count will be 2. When an object has some count, it is being used in the program and hence garbage collector will not remove it from memory. When an object is found with a reference count 0, garbage collector will understand that the object is not used by the program and hence it can be deleted from memory. Hence, the memory allocated for that object is deallocated or freed.

Frozen Binaries

When a software is developed in Python, there are two ways to provide the software to the end user. The first way is to provide the .pyc files to the user. The user will install PVM in his computer and run the byte code instructions of the .pyc files.

The other way is to provide the .pyc files, PVM along with necessary Python library. In this method, all the .pyc files, related Python library and PVM will be converted into a single executable file (generally with .exe extension) so that the user can directly execute that file by double clicking on it. In this way, converting the Python programs into true executables is called *frozen binaries*. But frozen binaries will have more size than that of simple .pyc files since they contain PVM and library files also.

For creating Frozen binaries, we need to use other party softwares. For example, py2exe is a software that produces frozen binaries for Windows operating system. We can use pyinstaller for UNIX or LINUX. Freeze is another program from Python organization to generate frozen binaries for UNIX.

RUNNING A PYTHON PROGRAM

Running a Python program can be done from 3 environments: 1. Command line window 2. IDLE graphics window 3. System prompt

Go get help, type help().
Type topics, FUNCTIONS, modules
Press <ENTER> to quit.
In IDLE window, click on help -> 'Python Docs' or F1 button to get documentation help.
Save a Python program in IDLE and reopen it and run it.

To see the .pyc file:
```
C:\>python –m py_compile first.py
```

To view byte code:
```
C:\> python –m dis first.py
```

PEP (Python Enhancement Proposal)

LEARNING RESOURCE ON INTERNET

www.tutorialspoint.com

COMMENTS (2 types)

# single line comments
“”” or ‘’’ multi line comments

Docstrings

If we write strings inside “”” or ‘’’ and if these strings are written as first statements in a module, function, class or a method, then these strings are called *documentation strings* or *docstrings*. These docstrings are useful to create an API documentation file from a Python program. An API (Application Programming Interface) documentation file is a text file or html file that contains description of all the features of a software, language or a product.

DATATYPES

A datatype represents the type of data stored into a variable (or memory).

Built-in datatypes

The built-in datatypes are of 5 types:

- None Type
- Numeric types
- Sequences
- Sets
- Mappings

None type: an object that does not contain any value.
Numeric types: int, float, complex.
Boolean type: bool.
Sequences: str, bytes, bytearray, list, tuple, range.

int type: represents integers like 12, 100, -55.
float type: represents float numbers like 55.3, 25e3.
complex type: represents complex numbers like 3+5j or 3-10.5J. Complex numbers will be in the form of a+bj or a+bJ. Here ‘a’ is called real part and ‘b’ is called ‘imaginary part’ and ‘j’ or ‘J’ indicates √-1.

NOTE:
Binary numbers are represented by a prefix 0b or 0B. Ex: 0b10011001
Hexadecimal numbers are represented by a prefix 0x or 0X. Ex: 0X11f9c
Octal numbers are represented by a prefix 0o or 0O. Ex: 0o145.

bool type: represents any of the two boolean values, True or False.
Ex: a = 10>5 # here a is treated as bool type variable.
     print(a) #displays True

NOTE:

1. To convert a float number into integer, we can use int() function. Ex: int(num)
2. To convert an integer into float, we can use float() function.
3. bin() converts a number into binary. Ex: bin(num)
4. oct() converts a number into octal.
5. hex() converts a number into hexadecimal.

STRINGS

str datatype: represents string datatype. A string is enclosed in single quotes or double quotes.
Ex: s1 = "Welcome"
     s2 = 'Welcome'

A string occupying multiple lines can be inserted into triple single quotes or triple double quotes.
Ex: s1 = ''' This is a special training on
                  Python programming that
                  gives insights into Python language.
             '''
To display a string with single quotes.
Ex: s2 = """This is a book 'on Core Python' programming"""

To find length of a string, use len() function.
Ex: s3 = 'Core Python'
n = len(s3)
print(n) -> 11

We can do indexing, slicing and repetition of strings.
Ex: s = "Welcome to Core Python"
print(s)   -> Welcome to Core Python
print(s[0]) -> W
print(s[0:7]) -> Welcome
print(s[:7]) -> Welcome
print(s[1:7:2]) -> ecm
print(s[-1] -> n
print(s[-3:-1]) -> ho
print(s[1]*3) -> eee
print(s*2) ->Welcome to CorePython Welcome to CorePython

Remove spaces using rstrip(), lstrip(), strip() methods.
Ex: name = " Vijay Kumar "
print(name.strip())

We can find substring position in a string using find() method. It returns -1 if not found.

Ex: n = str.find(sub, 0, len(str))

We can count number of substrings in a string using count() method. Returns 0 if not found.
Ex: n = str.count(sub)

We can replace a string s1 with another string s2 in a main string using replace() method.
Ex: str.replace(s1, s2)

We can change the case of a string using upper(), lower(), title() methods.
Ex: str.upper()

CHARACTERS

There is no datatype to represent a single character in Python. Characters are part of **str** datatype.

Ex:
```
str = "Hello"
print(str[0])
H

for i in str: print(i)

H
e
l
l
o
```

BYTES AND BYTEARRAY

bytes datatype: represents a group of positive integers in the range of 0 to 255 just like an array. The elements of bytes type cannot be modified.
Ex:
```
arr = [10, 20, 55, 100, 99]
x = bytes(arr)
for i in x:
   print(i)

10
20
55
100
99
```

bytearray datatype: same as bytes type but its elements can be modified.
```
arr = [10,20,55,100,99]
x=bytearray(arr)
x[0]=11
x[1]=21
for i in x: print(i)
```

```
11
21
55
100
99
```

NOTE:
We can do only indexing in case of bytes or bytearray datatypes. We cannot do slicing or repetitions.

LISTS

A **list** is similar to an array that can store a group of elements. A list can store different types of elements and can grow dynamically in memory. A list is represented by square braces [ ]. List elements can be modified.
Ex:

```
lst = [10, 20, 'Ajay', -99.5]
print(lst[2])
Ajay
```

To create an empty list.

```
lst = []    # then we can append elements to this list as lst.append('Vinay')
```

NOTE:
Indexing, slicing and repetition are possible on lists.

```
print(lst[1])
20

print(lst[-3:-1])
[20, 'Ajay']

lst = lst*2
print(lst)
[10, 20, 'Ajay', -99.5, 10, 20, 'Ajay', -99.5]
```

We can use len() function to find the no. of elements in the list.

```
n = len(lst)  -> 4
```

del() function is for deleting an element at a particular position.

```
del(lst[1]) -> deletes 20
```

remove() will remove a particular element. clear() wil delete all elements from the list.

```
lst.remove('Ajay')
lst.clear()
```

We can update the list elements by assignment.

```
lst[0] = 'Vinod'
lst[1:3] = 10, 15
```

max() and min() functions return the biggest and smallest elements.

```
max(lst)
min(lst)
```

Other methods of lists

Example	Description
lst.index(x)	Returns the first occurrence of x in lst.
lst.count(x)	Returns the number of occurrences of x in lst.
lst.insert(i, x)	Inserts x into lst in the position specified by i.
lst.copy()	Copies all elements of lst into a new list and returns it.
lst.extend(lst1)	Appends lst1 to lst.
lst.sort()	Sorts all elements into ascending order.
lst.sort(reverse=True)	Sorts all elements into descending order.
lst.reverse()	Reverses the sequence of elements.

TUPLES

A tuple is similar to a list but its elements cannot be modified. A tuple is represented by parentheses ( ).
Indexing, slicing and repetition are possible on tuples also.

Ex:

```
tpl=( )   # creates an empty tuple
tpl=(10, )  # with only one element – comma needed after the element

tpl = (10, 20, -30, "Raju")
print(tpl)
(10, 20, -30, 'Raju')
tpl[0]=-11   # error

print(tpl[0:2])
(10, 20)

tpl = tpl*2
print(tpl)
(10, 20, -30, 'Raju', 10, 20, -30, 'Raju')
```

NOTE: len(), count(), index(), max(), min() functions are same in case of tuples also.
We cannot use append(), extend(), insert(), remove(), clear() methods on tuples.

To sort the elements of a tuple, we can use sorted() method.

```
sorted(tpl)   # sorts all elements into ascending order
sorted(tpl, reverse=True)  # sorts all elements into descending order
```

To convert a list into tuple, we can use tuple() method.

```
tpl = tuple(lst)
```

RANGE DATATYPE

range represents a sequence of numbers. The numbers in the range cannot be modified. Generally, range is used to repeat a for loop for a specified number of times.

Ex: we can create a range object that stores from 0 to 4 as:
```
r = range(5)

print(r[0])  -> 0

for i in r: print(i)

0
1
2
3
4
```

Ex: we can also mention step value as:
```
r = range(0, 10, 2)
for i in r: print(i)

0
2
4
6
8

r1 = range(50, 40, -2)
for i in r1: print(i)

50
48
46
44
42
```

SETS

A **set datatype** represents unordered collection of elements. A set does not accept duplicate elements where as a list accepts duplicate elements. A set is written using curly braces { }. Its elements can be modified.

Ex:
```
s = {1, 2, 3, "Vijaya"}
print(s)
{1, 2, 3, 'Vijaya'}
```

NOTE: Indexing, slicing and repetition are not allowed in case of a set.

To add elements into a set, we should use update() method as:

```
s.update([4, 5])
print(s)
{1, 2, 3, 4, 5, 'Vijaya'}
```

To remove elements from a set, we can use remove() method as:

```
s.remove(5)
print(s)
{1, 2, 3, 4, 'Vijaya'}
```

A **frozenset** datatype is same as set type but its elements cannot be modified.
Ex:

```
s = {1, 2, -1, 'Akhil'}   -> this is a set
s1 = frozenset(s)   -> convert it into frozenset
for i in s1: print(i)

1
2
Akhil
-1
```

NOTE: update() or remove() methods will not work on frozenset.

MAPPING DATATYPES

A map indicates elements in the form of key – value pairs. When key is given, we can retrieve the associated value. A **dict** datatype (dictionary) is an example for a ‘map’.

```
d = {10: 'kamal', 11:'Subbu', 12:'Sanjana'}
print(d)
{10: 'kamal', 11: 'Subbu', 12: 'Sanjana'}
```

keys() method gives keys and values() method returns values from a dictionary.

```
k = d.keys()
for i in k: print(i)
10
11
12

for i in d.values(): print(i)

kamal
Subbu
Sanjana
```

To display value upon giving key, we can use as:
Ex: d = {10: 'kamal', 11:'Subbu', 12:'Sanjana'}

d[10] gives ‘kamal’

To create an empty dictionary, we can use as:
d = {}

Later, we can store the key and values into d, as:

d[10] = ‘Kamal’
d[11] = ‘Pranav’
We can update the value of a key, as: d[key] = newvalue.
Ex: d[10] = ‘Subhash’

We can delete a key and corresponding value, using del function.
Ex: del d[11] will delete a key with 11 and its value also.

PYTHON AUTOMATICALLY KNOWS ABOUT THE DATATYPE

The datatype of the variable is decided depending on the value assigned. To know the datatype of the variable, we can use type() function.
Ex:

```
x = 15  #int type
print(type(x))
<class 'int'>
 x = 'A'  #str type
print(type(x))
<class 'str'>
 x = 1.5  #float tye
print(type(x))
<class 'float'>
x = "Hello"  #str type
print(type(x))
<class 'str'>
x = [1,2,3,4]
print(type(x))
<class 'list'>
x = (1,2,3,4)
print(type(x))
<class 'tuple'>
x = {1,2,3,4}
print(type(x))
<class 'set'>
```

Literals in Python

A *literal* is a constant value that is stored into a variable in a program.
a = 15
Here, ‘a’ is the variable into which the constant value ‘15’ is stored. Hence, the value 15 is called ‘literal’. Since 15 indicates integer value, it is called ‘integer literal’.

Ex: a = 'Srinu' → here 'Srinu' is called string literal.
Ex: a = True → here, True is called Boolean type literal.

User-defined datatypes

The datatypes which are created by the programmers are called 'user-defined' datatypes. For example, an array, a class, or a module is user-defined datatypes. We will discuss about these datatypes in the later chapters.

Constants in Python

A constant is similar to variable but its value cannot be modified or changed in the course of the program execution. For example, pi value 22/7 is a constant. Constants are written using caps as PI.

Identifiers and Reserved words

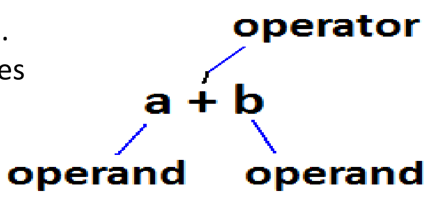
An identifier is a *name* that is given to a variable or function or class etc. Identifiers can include letters, numbers, and the underscore character (_). They should always start with a nonnumeric character. Special symbols such as ?, #, $, %, and @ are not allowed in identifiers. Some examples for identifiers are salaray, name11, gross_income, etc.

Reserved words are the words which are already reserved for some particular purpose in the Python language. The names of these reserved words should not be used as identifiers. The following are the reserved words available in Python:

and	del	from	nonlocal	try
as	elif	global	not	while
assert	else	if	or	with
break	except	import	pass	yield
class	exec	in	print	False
continue	finally	is	raise	True
def	for	lambda	return	

OPERATORS

A symbol that performs an operation.
An operator acts on variables or values that are called 'operands'.



Arithmetic operators
They perform basic arithmetic operations.

a=13, b = 5

Operator	Meaning	Example	Result
+	Addition operator.	a+b	18

-	Subtraction operator.	a-b	8
*	Multiplication operator.	a*b	65
/	Division operator.	a/b	2.6
%	Modulus operator. Gives remainder of division.	a%b	3
**	Exponent operator. a ** b gives the value of a to the power of b.	a**b	371293
//	Integer division. This is also called Floor division. Performs division and gives only integer quotient.	a//b	2

Assignment operators
To assign right side value to a left side variable.

Operator	**Example**	**Meaning**
=	z = x+y	Assignment operator. i.e. x+y is stored into z.
+=	z+=x	Addition assignment operator. i.e. z = z+x.
-=	z-=x	Subtraction assignment operator. i.e. z = z-x.
*=	z*=x	Multiplication assignment operator. i.e. z = z *x.
/=	z/=x	Division assignment operator. i.e. z = z/x.
%=	z%=x	Modulus assignment operator. i.e. z = z%x.
**=	z**=y	Exponentiation assignment operator. i.e. z = z**y.
//=	z//=y	Floor division assignment operator. i.e. z = z// y.

Ex:
```
a=b=c=5
print(a,b,c)
5 5 5

a,b,c=1,2,'Hello'
print(a,b,c)
1 2 Hello

x = [10,11,12]
a,b,c = 1.5, x, -1
print(a,b,c)
1.5 [10, 11, 12] -1
```
Unary minus operator
Converts +ve value into negative and vice versa.

Relational operators
Relational operators are used to compare two quantities. They return either True or False (bool datatype).

Ex:
```
a, b = 1, 2
print(a>b)
False
```

Ex:
1<2<3<4 will give True
1<2>3<4 will give False

Logical operators
Logical operators are useful to construct compound conditions. A compound condition is a combination of more than one simple condition. 0 is False, any other number is True.

X=1, y=2

Operator	Example	Meaning	Result
and	x and y	And operator. If x is False, it returns x, otherwise it returns y.	2
or	x or y	Or operator. If x is False, it returns y, otherwise it returns x.	1
not	not x	Not operator. If x is False, it returns True. If x is True it returns False.	False

Ex:
```
x=1; y=2; z=3
if(x<y or y>z):
        print('Yes')
else:
        print('No')   -> displays Yes
```

Boolean operators
Boolean operators act upon 'bool' type values and they provide 'bool' type result. So the result will be again either True or False.

x = True, y = False

Operator	Example	Meaning	Result
and	x and y	Boolean and operator. If both x and y are True, then it returns True, otherwise False.	False
or	x or y	Boolean or operator. If either x or y is True, then it returns True, else False.	True
not	not x	Boolean not operator. If x is True, it returns False, else True.	False

INPUT AND OUTPUT

print() function for output

Example	Output
print()	Blank line
print("Hai")	Hai
print("This is the \nfirst line")	This is the first line
print("This is the \\nfirst line")	This is the \nfirst line
print('Hai'*3)	HaiHaiHai
print('City=', "Hyderabad")	City= Hyderabad

print(‘City=’+”Hyderabad”)	City=Hyderabad
print(a, b)	1 2
print(a, b, sep=”,”)	1,2
print(a, b, sep=’-----‘)	1-----2

print("Hello")
print("Dear")
	Hello
	Dear

print("Hello", end='')
print("Dear", end='')
	HelloDear

a=2
print('You typed ', a, 'as input')
	
	You typed 2 as input

%i, %f, %c, %s can be used as format strings.
name='Linda'; sal=12000.50
print('Hai', name, 'Your salary is', sal)
print('Hai %s, Your salary is %.2f' % (name, sal))
	
	Hai Linda Your salary is 12000.5
	Hai Linda, Your salary is 12000.50

print('Hai {}, Your salary is {}'.format(name, sal))
print('Hai {0}, Your salary is {1}'.format(name, sal))
print('Hai {1}, Your salary is {0}'.format(name, sal))
	Hai Linda, Your salary is 12000.5
	Hai Linda, Your salary is 12000.5
	Hai 12000.5, Your salary is Linda

input() function for accepting keyboard input

Example
str = input()
str = input(‘Enter your name= ‘)
a = int(input(‘Enter int number: ‘)
a = float(input(‘Enter a float number: ‘)
a,b,c = [int(x) for x in input("Enter three numbers: ").split()]
a,b,c = [int(x) for x in input('Enter a,b,c: ').split(',')]
a,b,c = [x for x in input('Enter 3 strings: ').split(',')]
lst = [float(x) for x in input().split(',')]
lst = eval(input(‘Enter a list: ‘))

PROGRAMS
1. Enter id number, name and salary one by one and display them.
2. Enter 3 numbers and find their sum and average.
3. Enter the radius of a circle and find the area.
4. Enter a character and display it.
5. Enter two strings and display the total string.
6. Enter a number and display its cube value.
7. Sort a group of strings into ascending order.
8. Enter a list of different elements and display them.

CONTROL STATEMENTS

Control the flow of execution

- if statement
- if…else statement

- if…elif…else statement
- while loop
- for loop
- break statement
- continue statement
- pass statement
- return statement

if statement
executes statements based on a condition.

```
if condition:
   statements
```

if…else statement
executes statements1 else statements2 based on a condition.

```
if condition:
   statements1
else:
   statements2
```

PROGRAMS
9. Test whether a given number is even or odd.

if…elif…else statement
executes statements depending on multiple conditions.

```
if condition1:
   statements1
elif condition2:
   statements2
elif condition3:
   statements3
else:
   statements4
```

PROGRAMS
10. Test whether a given number is even or odd or zero.

while loop
executes statements as long as the condition is True.

```
while condition:
   statements
```

PROGRAMS
11. Display numbers from 1 to 10.
12. Display even numbers between m and n.

for loop
to iterate over the elements of a sequence like string, list, tuple, set, range, etc.

```
for var in sequence:
   statements
```

PROGRAMS
13. Display numbers from 100 to 110.
14. Display even numbers from 100 to 110.
15. Display the individual letters of a string.
16. Find sum of list of numbers.
17. Display required multiplication table.

else suit
else can be used with for loop or while loop. The statements in the else part are always executed.

```
for var in sequence:
   statements
else:
   statements

while condition:
   statements
else:
   statements
```

PROGRAMS
18. Enter a number and find out if it exists in a list.

break statement
to come out of a for loop or while loop.

continue statement
to continue the next iteration of a loop.

PROGRAMS
19. To break a loop of 1 to 10 at 5.
20. Do not display a loop from 5 onwards.

pass statement
does nothing. It is used in ‘if statement’ or a ‘loop’ to represent no operation.

PROGRAMS
21. Retrieve only negative numbers from a list.

return statement
to return result from a function.

PROGRAMS
22. A function to return the sum of two numbers from a function.

assert statement
to check if a particular condition is fulfilled or not. If the condition fails, it displays the assert statement and also raises AssertionError.

assert condition, “message”

PROGRAMS
23. Check if the user entered a number greater than 10 or not.

COMMAND LINE ARGUMENTS

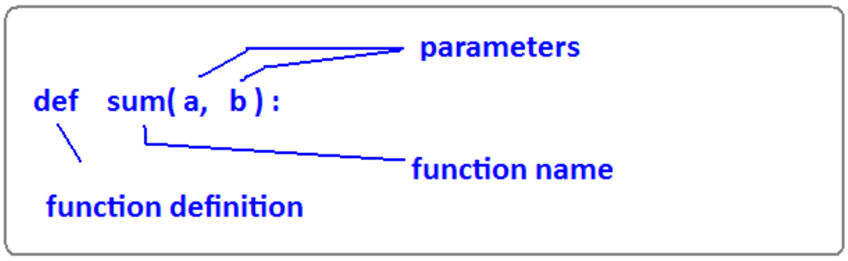
The arguments passed to a Python program at command prompt. They are stored in the form of a list in argv defined in ‘sys’ module.
len(sys.argv) gives number of command line args.
sys.argv is a list that contains all command line args.
sys.argv[0] contains the program name.

PROGRAMS
24. Display the no. of command line args and their names.
25. Enter a group of numbers at command line and find their sum and average.

FUNCTIONS

Contains a group of statements to perform a particular task. A function written inside a class is a ‘method’. A method is called as: obj.methodname() or class.methodname().

Function definition = function name, parameters and body.



```
# function definition
def sum(a, b) :
    """ to find sum of two numbers """
    c = a + b
    print('Sum= ', c)

# call the function
sum(5, 10)

# call again
sum('Hello', 'Dear')
```

: indicates beginning of body.
docstring.
function body

PROGRAMS
26. A function to find sum of two numbers.

A function can return more than one value. When several values are returned, they are stored into a tuple and then returned.

PROGRAMS
27. A function to return result of addition and subtraction of two numbers.
28. A function to return results of addition, subtraction, multiplication and division.

It is possible to send a group of strings or numbers in the form of a list to a function and do any operation inside the function.

PROGRAMS
29. Enter a group of numbers from keyboard and sort them using a function.

LOCAL AND GLOBAL VARIABLES
A variable declared inside a function is local variable. Local variable is available only within that function. A variable declared above the function becomes global variable. It is available in the function and also outside of the function.

PROGRAMS
30. Create a global variable and local variable and display their values.

When local variable and global variable have same name, we can use globals() function to retrieve the global variable value inside the function. Ex: globals()['var']

PROGRAMS
31. Use globals() to retrieve the global variable inside a function.

Functions are first class objects

That means we can use functions exactly as objects.

- It is possible to assign a function to a variable.
- It is possible to define one function inside another function.
- It is possible to pass a function as parameter to another function.
- It is possible that a function can return another function.

PROGRAMS
32. Assign a function to a variable.
33. Define message() function inside display() function.
34. Pass message() function to display() function as parameter.
35. Return message() function from display() function.

Recursive functions
A function calling itself repeatedly.

PROGRAMS
36. Using recursion, calculate factorial value of a given number.

Formal and actual arguments

When a function is defined, it may have some parameters. These parameters are useful to receive values from outside of the function. They are called 'formal arguments'. When we call the function, we should pass data or values to the function. These values are called 'actual arguments'. In the following code, 'a' and 'b' are formal arguments and 'x' and 'y' are actual arguments.

```
def sum(a, b):  # a, b are formal arguments
   c = a+b
   print(c)

# call the function
x=10; y=15
sum(x, y)   # x, y are actual arguments
```

The actual arguments used in a function call are of 4 types:

- Positional arguments
- Keyword arguments
- Default arguments
- Variable length arguments

Positional arguments

These are the arguments passed to a function in correct positional order. Here, the number of arguments and their positions in the function definition should match exactly with the number and position of the argument in the function call

```
def attach(s1, s2):  # function definition
attach('New', 'York')  # positional arguments
```

Keyword arguments

Keyword arguments are arguments that identify the parameters by their names.

```
def grocery(item, price):  # function definition
grocery(item='Sugar', price=50.75)  # key word arguments
```

Default arguments

We can mention some default value for the function parameters in the definition.

```
def grocery(item, price=40.00):  # default argument is price
grocery(item='Sugar')   # default value for price is used
```

Variable length arguments

A variable length argument is an argument that can accept any number of values. The variable length argument is written with a ‘ * ‘ symbol before it in the function definition, as:

```
def add(farg, *args):   # *args can take 0 or more values
add(5, 10)
add(5, 10, 20, 30)
```

Here, ‘farg’ is the formal argument and ‘*args’ represents variable length argument. We can pass 1 or more values to this ‘*args’ and it will store them all in a tuple.

Function decorators

A decorator is a function that accepts a function as parameter and returns a function. A decorator takes the result of a function, modifies the result and returns it. Thus decorators are useful to perform some additional processing required by a function.

1. We should define a decorator function with another function name as parameter.

```
def decor(fun):
```

2. We should define a function inside the decorator function. This function actually modifies or decorates the value of the function passed to the decorator function.

```
def decor(fun):
   def inner():
      value = fun()   # access value returned by fun()
      return value+2   # increase the value by 2
   return inner   # return the inner function
```

3. Return the inner function that has processed or decorated the value. In our example, in the last statement, we were returning inner() function using return statement. With this, the decorator is completed.

The next question is how to use the decorator. Once a decorator is created, it can be used for any function to decorate or process its result. For example, let us take num() function that returns some value, e.g. 10.

```
def num():
   return 10
```

Now, we should call decor() function by passing num() function name as:

```
result_fun = decor(num)
```

So, ‘result_fun’ indicates the resultant function. Call this function and print the result, as:

print(result_fun())

PROGRAMS
37. Write a decorator function to increase the value by 2.

Generators

Generators are functions that return a sequence of numbers. A generator function is written like an ordinary function but it uses 'yield' statement. This statement is useful to return the value.

PROGRAMS
38. Create a generator object with numbers from 5 to 10.

Creating our own modules in Python

A module represents a group of classes, methods, functions and variables. In Python, we have several built-in modules like sys, io, time etc. Just like these modules, we can also create our own modules and use them whenever we need them. Once a module is created, any programmer in the project team can reuse that module. Hence, modules will make software development easy and faster.

PROGRAMS
39. Create 'mymodule' with addition and subtraction functions and use them in another program.

The special variable __name__

When a program is executed in Python, there is a special variable internally created by the name '__name__'. This variable stores information regarding whether the program is executed as an individual program or as a module. When the program is executed directly, the Python interpreter stores the value '__main__' into this variable. When the program is imported as a module into another program, then Python interpreter stores the module name into this variable. Thus, by observing the value of the variable '__name__', we can understand how the program is executed.

PROGRAMS
40. Create 'one' module with display() function. Find out how it is executed.

Pacakges and Libraries

A package is a folder that contains a group of modules. To distinguish between a normal folder and package, we need to store an empty file by the name '__init__.py' in the package folder.

A Library is a group of packages. The main advantage of libraries and packages is to reuse their contents by any programmer in different projects.

LAMBDAS OR ANONYMOUS FUNCTIONS

A function without a name is called 'anonymous function'. So far, the functions we wrote were defined using the keyword 'def'. But anonymous functions are not defined using 'def'. They are defined using the keyword *lambda* and hence they are also called ‘Lambda functions’. Let us take a normal function that returns square of a given value.

```
def square(x):
   return x*x
```

The same function can be written as anonymous function as:

```
lambda x: x*x
```

Observe the keyword ‘lambda’. This represents that an anonymous function is being created. After that we have written an argument of the function, i.e. ‘x’. Then colon (:) represents the beginning of the function that contains an expression x * x. Please observe that we did not use any name for the function here. So, the format of lambda functions is:

```
lambda argument_list : expression
```

Normally, if a function returns some value, we assign that value to a variable, as:

```
y = square(5)
```

But, lambda functions return a function and hence they should be assigned to a function, as:

```
f = lambda x: x*x
```

Here, ‘f’ is the function name to which the lambda expression is assigned. Now, if we call the function f(), as:

```
value = f(5)
```

Now, ‘value’ contains the square value of 5, i.e. 25.

PROGRAMS
41. Create a lambda function to find square value of a number.
42. Write a lambda to test whether a given number is even or not.

Using lambdas with filter() function

The filter() function is useful to filter out the elements of a sequence depending on the result of a function. We should supply a function and a sequence to the filter() function as:

```
filter(function, sequence)
```

Here, the ‘function’ represents a function name that may return either True or False; and ‘sequence’ represents a list, string or tuple.

PROGRAMS
43. Create a lambda that returns even numbers from a list of numbers.

Using lambdas with map() function

The map() function is similar to filter() function but it acts on each element of the sequence and perhaps changes the elements. The format of map() function is:

map(function, sequence)

The ‘function’ performs a specified operation on all the elements of the sequence and the modified elements are returned which can be stored in another sequence.

PROGRAMS
44. Create a lambda that returns squares of all elements in a list.

Using lambdas with reduce() function

reduce() function reduces a sequence of elements to a single value by processing the elements according to a function supplied. The reduce() function is used in the format:

reduce(function, sequence)

NOTE: reduce() function belongs to ‘functools’ module.

PROGRAMS
45. Create a lambda function to calculate products of elements of a list.

ARRAYS

To work with arrays, we use numpy (numerical python) package.
For complete help on numpy: https://docs.scipy.org/doc/numpy/reference/

An array is an object that stores a group of elements (or values) of same datatype. Array elements should be of same datatype. Arrays can increase or decrease their size dynamically.

NOTE: We can use for loops to display the individual elements of the array.

To work with numpy, we should import that module, as:
import numpy
import numpy as np
from numpy import *

Single dimensional (or 1D) arrays

A 1D array contains one row or one column of elements. For example, the marks of a student in 5 subjects.

Creating single dimensional arrays

Creating arrays in numpy can be done in several ways. Some of the important ways are:

- Using array() function
- Using linspace() function
- Using logspace() function

- Using arange() function
- Using zeros() and ones() functions.

Creating 1D array using array()

To create a 1D array, we should use array() method that accepts list of elements.
Ex: arr = numpy.array([1,2,3,4,5])

PROGRAMS
46. Create different types of 1D arrays using numpy.

Creating 1D array using linspace()

linspace() function is used to create an array with evenly spaced points between a starting point and ending point. The form of the linspace() is:

linspace(start, stop, n)

'start' represents the starting element and 'stop' represents the ending element. 'n' is an integer that represents the number of parts the elements should be divided. If 'n' is omitted, then it is taken as 50. Let us take one example to understand this.
a = linspace(0, 10, 5)

In the above statement, we are creating an array 'a' with starting element 0 and ending element 10. This range is divided into 5 equal parts and hence the points will be 0, 2.5, 5, 7.5 and 10. These elements are stored into 'a'. Please remember the starting and ending elements 0 and 10 are included.

PROGRAMS
47. Create an array with 5 equal points using linspace().

Creating arrays using logspace

logspace() function is similar to linspace(). The linspace() produces the evenly spaced points. Similarly, logspace() produces evenly spaced points on a logarithmically spaced scale. logspace is used in the following format:

logspace(start, stop, n)

The logspace() starts at a value which is 10 power of 'start' and ends at a value which is 10 power of 'stop'. If 'n' is not specified, then its value is taken as 50. For example, if we write:

a = logspace(1, 4, 5)

This function represents values starting from 10^1 to 10^4. These values are divided into 5 equal points and those points are stored into the array 'a'.

PROGRAMS
48. A program to create an array using logspace().

Creating 1D arrays using arange() function

The arange() function in numpy is same as range() function in Python. The arange() function is used in the following format:

arange(start, stop, stepsize)

This creates an array with a group of elements from 'start' to one element prior to 'stop' in steps of 'stepsize'. If the 'stepsize' is omitted, then it is taken as 1. If the 'start' is omitted, then it is taken as 0. For example,

arange(10)

will produce an array with elements 0 to 9.

arange(5, 10, 2)

will produce an array with elements: 5,7,9.

PROGRAMS

49. A program to create an array with even number up to 10.

Creating arrays using zeros() and ones() functions

We can use zeros() function to create an array with all zeros. The ones() function is useful to create an array with all 1s. They are written in the following format:

zeros(n, datatype)
ones(n, datatype)

where 'n' represents the number of elements. we can eliminate the 'datatype' argument. If we do not specify the 'datatype', then the default datatype used by numpy is 'float'. See the examples:

zeros(5)

This will create an array with 5 elements all are zeros, as: [0. 0. 0. 0. 0.]. If we want this array in integer format, we can use 'int' as datatype, as:
zeros(5, int)

this will create an array as: [0 0 0 0 0].

If we use ones() function, it will create an array with all elements 1. For example,

ones(5, float)

will create an array with 5 integer elements all are 1s as: [1. 1. 1. 1. 1.].

Arithmetic operations on arrays

Taking an array as an object, we can perform basic operations like +, -, *, /, // and % operations on each element.
Ex:

```
import numpy
arr = numpy.array([10, 20, 30])
arr+5
```

Important Mathematical functions in numpy

Function	Meaning
concatenate([a, b])	Joins the arrays a and b and returns the resultant array.
sqrt(arr)	Calculates square root value of each element in the array ‘arr’.
power(arr, n)	Returns power value of each element in the array ‘arr’ when raised to the power of ‘n’.
exp(arr)	Calculates exponentiation value of each element in the array ‘arr’.
sum(arr)	Returns sum of all the elements in the array ‘arr’.
prod(arr)	Returns product of all the elements in the array ‘arr’.
min(arr)	Returns smallest element in the array ‘arr’.
max(arr)	Returns biggest element in the array ‘arr’.
mean(arr)	Returns mean value (average) of all elements in the array ‘arr’.
median(arr)	Returns median value of all elements in the array ‘arr’.
std(arr)	Gives standard deviation of elements in the array ‘arr’.
argmin(arr)	Gives index of the smallest element in the array. Counting starts from 0.
argmax(arr)	Gives index of the biggest element in the array. Counting starts from 0.
unique(arr)	Gives an array that contains unique elements of the array ‘arr’.
sort(arr)	Gives an array with sorted elements of the array ‘arr’ in ascending order.

Ex:

```
numpy.sort(arr)
numpy.max(arr)
numpy.sqrt(arr)
```

Aliasing the arrays

If ‘a’ is an array, we can assign it to ‘b’, as:

```
b = a
```

This is a simple assignment that does not make any new copy of the array ‘a’. It means, ‘b’ is not a new array and memory is not allocated to ‘b’. Also, elements from ‘a’ are not copied into ‘b’ since there is no memory for ‘b’. Then how to understand this assignment statement? We should understand that we are giving a new name ‘b’ to the same array referred by ‘a’. It means the names ‘a’ and ‘b’ are referencing same array. This is called ‘aliasing’.

‘Aliasing’ is not ‘copying’. Aliasing means giving another name to the existing object. Hence, any modifications to the alias object will reflect in the existing object and vice versa.

Viewing and Copying arrays

We can create another array that is same as an existing array. This is done by view() method. This method creates a copy of an existing array such that the new array will also contain the same elements found in the existing array. The original array and the newly created arrays will share different memory locations. If the newly created array is modified, the original array will also be modified since the elements in both the arrays will be like mirror images.

We can create a view of ‘a’ as:

```
b = a.view()
```

Viewing is nothing but copying only. But it is called ‘shallow copying’ as the elements in the view when modified will also modify the elements in the original array. So, both the arrays will act as one and the same. Suppose we want both the arrays to be independent and modifying one array should not affect another array, we should go for ‘deep copying’. This is done with the help of copy() method. This method makes a complete copy of an existing array and its elements. When the newly created array is modified, it will not affect the existing array or vice versa. There will not be any connection between the elements of the two arrays.

We can create a copy of ’a’ as:

```
b = a.copy()
```

PROGRAMS

50. Create a 1D array using arange() and then create another array by alias, view and copy.

Indexing and slicing in 1D arrays are possible

Ex:

```
import numpy as np
arr = np.array([10, 20, 30, 40, 50, 90])
arr[1]  -> 20

print(arr[1:5])
[20 30 40 50]

arr[1:5:2]
array([20, 40])

arr[:5:]
array([10, 20, 30, 40, 50])

arr[::]
array([10, 20, 30, 40, 50, 90])

arr[0::2]
```

```
array([10, 30, 50])

arr[5:0:-1]
array([90, 50, 40, 30, 20])
```

Attributes of an array

Numpy's array class is called *ndarray*. It is also known by alias name *array*. Let us remember that there is another class 'array' in Python that is different from numpy's 'array' class. This class contains the following important attributes (or variables):

ndim
'ndim' represents the number of dimensions or axes of the array. The number of dimensions is also referred to as 'rank'. For a single dimensional array, it is 1 and for a two dimensional array, it is 2. See the examples:

```
arr1 = array([1,2,3,4,5])  # 1D array
print(arr1.ndim) displays 1

arr2 = array([[1,2,3], [4,5,6]])  # 2D array with 2 rows and 3 elements
print(arr2.ndim) displays 2
```

shape
'shape' gives the shape of an array. The shape is a tuple listing the number of elements along each dimension. A dimension is called an axis. For a 1D array, shape gives the number of elements in the row. For a 2D array, it specifies the number of rows and columns in each row. We can also change the shape using 'shape' attribute. See the examples:

```
arr1 = array([1,2,3,4,5])
print(arr1.shape) displays (5,)   # no. of elements

arr2 = array([[1,2,3], [4,5,6]])
print(arr2.shape) displays (2, 3)  # 2 rows and 3 cols

arr2.shape = (3, 2)  # change shape of arr2 to 3 rows and 2 cols
print(arr2)  displays the following:
[[1  2]
  [3  4]
  [5  5]]
```

size
gives the total number of elements in the array. For example,

```
arr1 = array([1,2,3,4,5])
print(arr1.size) displays 5

arr2 = array([[1,2,3], [4,5,6]])
print(arr2.size) displays 6
```

itemsize
This gives the memory size of the array element in bytes. As we know, 1 byte is equal to 8 bits. For example,

```
arr1 = array([1,2,3,4,5])
print(arr1.itemsize)  displays 4

arr2 = array([1.1,2.1,3.5,4,5.0])
print(arr2.itemsize) displays 8
```

dtype
This attribute gives the datatype of the elements in the array. For example,

```
arr1 = array([1,2,3,4,5])  # integer type array
print(arr1.dtype) displays int32

arr2 = array([1.1,2.1,3.5,4,5.0])  # float type array
print(arr2.dtype) displays float64
```

nbytes
gives the total number of bytes occupied by an array. The total number of bytes = size of the array * item size of each element in the array. For example,

```
arr2 = array([[1,2,3], [4,5,6]])
print(arr2.nbytes) displays 24
```

Apart from the attributes discussed in the preceding sections, we can use reshape() and flatten() methods which are useful to convert the 1D array into a 2D array and vice versa.

reshape()
This method is useful to change the shape of an array. The new array should have the same number of elements as in the original array. For example,

```
arr1 = arange(10)  # 1D array with 10 elements
print(arr1) displays the following:
[0 1 2 3 4 5 6 7 8 9]

arr1 = arr1.reshape(2, 5)  # change the shape as 2 rows, 5 cols
print(arr1) displays the following:
[[0 1 2 3 4]
 [5 6 7 8 9]]

arr1 = arr1.reshape(5, 2) # change the shape as 5 rows, 2 cols
print(arr1)   displays the following:
[[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

flatten()
This method is useful to return a copy of the array collapsed into one dimension. For example, let us take a 2D array as:

```
arr1 = array([[1,2,3],[4,5,6]])
print(arr1) displays the following:
[[1 2 3]
 [4 5 6]]
```

By using flatten() method, we can convert this array into 1D array, as:

```
arr1 = arr1.flatten()
print(arr1) displays the following:
[1 2 3 4 5 6]
```

Multi-dimensional arrays (2D, 3D, etc)

They represent more than one row and more than one column of elements. For example, marks obtained by a group of students each in five subjects.

Creating multi-dimensional arrays

We can create multi dimensional arrays in the following ways:

- Using array() function
- Using ones() and zeroes() functions
- Using eye() function
- Using reshape() function discussed earlier

Using array() function

To create a 2D array, we can use array() method that contains a list with lists.

PROGRAMS
51. Create and display a 2D array using array() method.

ones() and zeros() functions

The ones() function is useful to create a 2D array with several rows and columns where all the elements will be taken as 1. The format of this function is:

```
ones((r, c), dtype)
```

Here, ‘r’ represents the number of rows and ‘c’ represents the number of columns. ‘dtype’ represents the datatype of the elements in the array. For example,

```
a = ones((3, 4), float)
```

will create a 2D array with 3 rows and 4 columns and the datatype is taken as float. If 'dtype' is omitted, then the default datatype taken will be 'float'. Now, if we display 'a', we can see the array as:

```
[[1.  1.  1.  1.]
  [1.  1.  1.  1.]
  [1.  1.  1.  1.]]
```

The decimal point after each element represents that the elements are float type.

Just like ones() function, we can also use zeros() function to create a 2D array with elements filled with zeros. Suppose, we write:

```
b = zeros((3,4), int)
```

Then a 2D array with 2 rows and 4 columns will be created where all elements will be 0s, as shown below:

```
[[0  0  0  0]
  [0  0  0  0]
  [0  0  0  0]]
```

eye() function

The eye() function creates a 2D array and fills the elements in the diagonal with 1s. The general format of using this function is:

```
eye(n, dtype=datatype)
```

This will create an array with 'n' rows and 'n' columns. The default datatype is 'float'. For example, eye(3) will create a 3x3 array and fills the diagonal elements with 1s as shown below:

```
a = eye(3)

[[1.  0.  0.]
  [0.  1.  0.]
  [0.  0.  1.]]
```

Indexing and slicing in 2D arrays

Ex:

```
import numpy as np
arr = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

arr[0] gives 0th row -> [1,2,3]
arr[1] gives 1st row -> [4,5,6]
arr[0,1] gives 0th row, 1st column element -> 2
arr[2,1] gives 2nd row, 1st column element -> 8

```
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
a
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

a[0:2, 0:3] -> 0th row to 1st row, 0th column to 2nd column

```
array([[1, 2, 3],
       [5, 6, 7]])
```

a[1:3, 2:] -> 1th row to 2nd row, 2nd column to last column

```
array([[ 7,  8],
       [11, 12]])
```

MATRICES IN NUMPY

A 2D array with m rows and n cols - is an example for a mXn matrix.

matrix() method is useful to convert a 2D array into a matrix.
matrix(2D array)
matrix('string that contains elements')

Ex:

```
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
m = np.matrix(a)
m
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
```

We can retrieve diagonal elements of the matrix using diagonal() method.

```
d = np.diagonal(m)
print(d)
[1 5 9]
```

We can find biggest and smallest elements in the matrix using max() and min() methods.

```
b = m.max()
print(b)
9

s = m.min()
print(s)
1
```

We can find sum and average of elements using sum() and mean() methods.

```
sum = m.sum()
print(sum)
45
avg = m.mean()
```

```
print(avg)
5.0
```

We can sort the elements into ascending order using sort() method.

```
arr = np.array([[3,4,2], [1,2,9]])
m = np.matrix(arr)
m
matrix([[3, 4, 2],
        [1, 2, 9]])

s = np.sort(m)   # sort the elements in each row
s
matrix([[2, 3, 4],
        [1, 2, 9]])
s1 = np.sort(m, axis=0)   # sort the elements in each column
 s1
matrix([[1, 2, 2],
        [3, 4, 9]])
```

We can find transpose of the matrix using transpose() method.

```
t = m.transpose()
t
matrix([[3, 1],
        [4, 2],
        [2, 9]])
```

Matrix addition, subtraction, multiplication, division can be done by using +, -, * and / operators. ***(DO NOT USE IDLE)***

PROGRAMS

52. Do matrix multiplication and division.
53. Accept matrix from keyboard as a string and display its transpose.
54. Rewrite the above program to accept the matrix in matrix form.

NESTED LISTS

A list within another list is called a *nested* list. We know that a list contains several elements. When we take a list as an element in another list, then that list is called a nested list. For example, we have two lists 'a' and 'b' as:

```
a = [80, 90]
b = [10, 20, 30, a]
```

Observe that the list 'a' is inserted as an element in the list 'b' and hence 'a' is called a nested list. Let us display the elements of 'b', now.

```
print(b)
[10, 20, 30, [80, 90]]
```

The last element [80, 90] represents a nested list. So, ‘b’ has 4 elements and they are:

```
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = [80, 90]
```

So, b[3] represents the nested list and if we want to display its elements separately, we can use a for loop as:

```
for x in b[3]:
   print(x)
```

PROGRAMS
55. Store names and marks as nested list. When a name is given, display marks.

LIST COMPREHENSIONS

List comprehensions represent creation of new lists from an iterable object (like a list, set, tuple, dictionary or range) that satisfy a given condition. List comprehensions contain very compact code usually a single statement that performs the task.

Example : We want to create a list with squares of integers from 1 to 10. We can write code as:

```
squares = []   # create empty list
for x in range(1, 11):   # repeat x values from 1 to 10
   squares.append(x**2)   # add squares of x to the list
```

The preceding code will create ‘squares’ list with the elements as shown below:

```
 [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

The previous code can be rewritten in a compact way, as:

```
squares = [x**2 for x in range(1, 11)]
```

This is called list comprehension. From this, we can understand that a list comprehension consists of square braces containing an expression (i.e. x**2). After the expression, a for loop and then zero or more if statements can be written. See the syntax:

```
[ expression for item1 in iterable1 if statement1
                  for item2 in iterable2 if statement2
                  for item3 in iterable3 if statement3 ... ]
```

Observe the square braces around these statements. They indicate that a list will be returned as a result of the list comprehension.

NOTE: It is possible to do list comprehension with tuples also.

PROGRAMS

56. Write a list comprehension to get even nos upto 10.
57. Add corresponding elements of two lists and store them in another list.
58. Retrieve common elements in two lists.
59. Retrieve only the first letter of each word in the list.

LIST OF PROGRAMS (CORE PYTHON)

```
# 1. to accept id, name and salary
id = int(input('Enter id: '))
name = input('Enter name: ')
sal = input('Enter salary: ')
print('Your id={}\nName={}\nSalary={}'.format(id,name,sal))
```

```
# 2. sum and average
a,b,c = [float(x) for x in input('Enter 3 numbers: ').split()]
sum = a+b+c
avg = sum/3
print("Sum= %i\nAverage= %.2f" % (sum, avg))
```

```
# 3.area of a circle - v1
r = float(input('Enter radius: '))
PI = 22/7
area = PI*r**2
print('Area= ', area)

'''
# area of a circle - v2
import math
r = float(input('Enter radius: '))
area = math.pi*r*r
print('Area= ', area)
'''
```

```
# 4. enter a char and display
str = input('Enter a char: ')
print('You entered: '+ str[0])
```

```
# 5. joining strings
str1, str2 = input('Enter two strings: ').split(',')
print("Total string: "+str1+str2)
```

```
# 6.cube value
num = float(input('Enter a number: '))
print('Cube= ', num**3)
```

```
# 7.sorting a group of strings
lst = []

n = int(input('How many strings? '))

for i in range(n):
    s = input('Enter string: ')
    lst.append(s)

lst.sort()

for i in lst:
    print(i)
```

```
# 8.enter a list and display using eval()
lst = eval(input('Enter a list: '))
print("List= ", lst)
for i in lst:
    print(i)
```

```
# 9.test whether a given number is even or odd
x = int(input('Enter a number: '))
if x % 2 == 0:
    print(x,"is even")
else:
    print(x,"is odd")
```

```
# 10.even or odd or zero
no = int(input('Enter a number: '))
if(no ==0): print(no, "is zero")
elif(no % 2 == 0): print(no,"is even")
else: print(no, "is odd")
```

```
# 11.display numbers from 1 to 10
x=1
while(x<=10):
    print(x)
    x+=1
print("End")
```

```
# 12.display evens between m and n
m, n = [int(i) for i in input('Enter start, stop: ').split()]
x = m  # start from min
if x % 2 !=0: x=m+1

while(x<=n):
    print(x)
    x+=2
```

```
# 13.to display numbers from 100 to 110
x = 100
for i in range(100, 111):
    print(i)
```

```
# 14.to display even numbers from 100 to 110
x = 100
for i in range(100, 111, 2):
    print(i)
```

```
# 15.to display letters of a string
str = "Hello"
for i in str:
    print(i)
```

```
# 16.to find sum of list of numbers
lst = [10, 20, -10, 5.5]
sum=0
for i in lst:
    sum+=i
print('Sum=', sum)
```

```
# 17.to display a multiplication table
x = int(input('Which table? '))

for i in range(1, 21):
    print(i, ' X ', x, ' = ', i*x)
```

```
# 18.whether an element exists in the list or not
lst = [1,2,3,4]
s = int(input('Which element to search? '))
for i in lst:
    if s == i:
        print('Found')
        break
else:
    print('Not found')
```

```
# 19.to break a loop at 5
x=0
while x<=10:
    x+=1
    if x==5:
        break
    print(x)
```

```
# 20.not to display from 5 onwards
x=0
while x<=10:
    x+=1
    if x>=5:
        continue
    print(x)
```

```
# 21.to retrieve only negatives
num = [1,2,-3,-4,5,-6]

for i in num:
    if(i>0):
        pass
    else:
        print(i)
```

```
# 22.a function to return sum value
# define the function
def sum(a, b):
    c = a+b
    return c

# call the function
res1 = sum(5, 10)
res2 = sum(1.5, 2.6)
print('Result1= ', res1)
print('Result2= ', res2)
```

```
# 23.understanding assert stmt
x = int(input('Enter a number: '))
assert x>10, "Wrong input entered"
print('U entered: ', x)
```

```
# 24.command line args
import sys

# find no. of args
n = len(sys.argv)
print('No of args= ', n)

# get all args
lst = sys.argv
print('They are: ')
for i in lst: print(i)

# show program name
print('Program name= ', sys.argv[0])
```

```
# 25.command line args
import sys

# find no. of args
n = len(sys.argv)
print('No of args= ', n)

# get args from argv[1] till end and add them
sum=0
lst = sys.argv
for i in range(1, n):
    sum+=float(lst[i])

print('Sum= ', sum)
print('Average= ', sum/n)
```

```
# 26.function definition
def sum(a, b) :
        """ to find sum of two numbers """
        c = a + b
        print('Sum= ', c)

# call the function
```

```
sum(5, 10)

# call again
sum('Hello', 'Dear')
```

```
# 27.function definition
def sum_sub(a, b) :
        """ to find result of addition
        and subtraction """
        c = a + b
        d = a- b
        return c, d

# call the function
c, d = sum_sub(5, 10)
print("Results = {} and {}".format(c, d))
```

```
# 28.function to return several values
def fun(n1, n2) :
        a = n1+n2
        b = n1-n2
        c = n1*n2
        d = n1/n2
        return a,b,c,d

# call the function
tpl = fun(5, 10)
print(tpl)

for i in tpl: print(i)
```

```
# 29.sorting a group of numbers
def sortnum(lst):
    lst.sort()
    for i in lst: print(i)

# take a group of numbers from keyboard
print('Enter numbers: ')
lst = [float(x) for x in input().split(',')]

# call the function
sortnum(lst)
```

```
# 30.local and global variables
a = 5  # global var
def fun():
    a = 6  # local var
    print(a) # displays local var

fun()
print(a)  # displays global var
```

```
# 31.local and global variables
a = 5  # global var
def fun():
    a = 6  # local var
    print(a) # displays local var

    x = globals()['a']
    print(x)  # displayes global var
```

```
fun()
print(a)  # displays global var
```

```
# 32.assign a function to a variable
def display(str):
    return 'Hai '+str

var = display('Krish')
print(var)
```

```
# 33.define a function inside another function
def display(str):
    def message():
        return 'How are U? '
    result = message()+str
    return result

var = display('Krish')
print(var)
```

```
# 34.functions can be passed as parameters to other functions
def display(fun):
    return 'How are U? '+ fun

def message():
    return 'Krish'

var = display(message())
print(var)
```

```
# 35.functions can return other functions
def display():
    def message():
        return 'How are U?'

    return message

# call display() function and it returns message() function
# in the following code, var refers to the name: message.
var = display()
print(var())
```

```
# 36.recursive function
def factorial(n):
    if n==0:
        result=1
    else:
        result=n*factorial(n-1)
    return result

n = int(input('Enter number: '))
print('Factorial = ', factorial(n))
```

```
# 37.a decorator to increase the value by 2
def decor(fun):
    def inner():
        value = fun()
        return value+2
    return inner

"""
# this is the function to which decorator should be applied
def num():
    return 10

result = decor(num)
print(result())

"""
@decor
def num():
    return 10

print(num())
```

```
# 38.generator that returns sequence from x to y
def mygen(x, y):
    while x<=y:
        yield x
        x+=1

# fill generator object with 5 and 10
g = mygen(5, 10)

# display all numbers in the generator
for i in g:
    print(i, end=' ')
```

```
# 39.a module with 2 functions - save this a mymodule.py
def add(a, b):
    print(a+b)

def sub(a, b):
    print(a-b)

# use mymodule.
import mymodule
x, y = [int(x) for x in input('Enter two nos: ').split()]
mymodule.add(x,y)
mymodule.sub(x,y)
```

```
# 40.python program to display a message. save this as one.py
def display():
    print('Hello Python!')

if __name__ == '__main__':
    display()   # call display function
    print('This code is run as a program')
else:
    print('This code is run as a module')

# in this program one.py is imported as a module.
import one
one.display()
```

```
# 41.a lambda function to calculate square value
f = lambda x: x*x   # write lambda function
value = f(5)   # call lambda function
print('Square of 5 = ', value)   # display result
```

```
# 42.a lambda function to test a number is even or not
f = lambda x: 'Yes' if x%2==0 else 'No'   # write lambda function
value = f(18)   # call lambda function
print(value)   # display result
```

```
# 43.a lambda function that returns even numbers from a list
lst = [10, 23, 45, 46, 70, 99]
lst1 = list(filter(lambda x: (x%2 == 0) , lst))
print(lst1)
```

```
# 44.a lambda function that returns squares of numbers of a list
lst = [10, 23, 45, 46, 70, 99]
lst1 = list(map(lambda x: x*x , lst))
print(lst1)
```

```
# 45.Lambda that returns products of elements of a list
from functools import *
lst = [1, 2, 3, 4, 5]
result = reduce(lambda x, y: x*y, lst)
print(result)
```

```
# 46.create different types of 1D arrays
import numpy
arr1 = numpy.array([1,2,3,4])
print(arr1)

arr2 = numpy.array([1.5, 2.5, -3.44])
print(arr2)

arr3 = numpy.array(['a','b','c','d'])
print(arr3)

arr4 = numpy.array(['anil', 'gopal', 'vikas'])
print(arr4)
```

```
# 47.creating an array using linspace()
from numpy import *

# divide 0 to 10 into 5 parts and take those points in the array
a = linspace(0, 10, 5)
print('a = ', a)
```

```
# 48.creating an array using logspace()
from numpy import *

# divide 1 power 10 to 4 power 10 into 5 equal parts
# and take those points in the array
a = logspace(1, 4, 5)

# display the elements of a
for i in a: print(i, end=', ')
```

```
# 49.creating an array with even numbers up to 10
from numpy import *

# create an array using arange() function
```

```
a = arange(2, 11, 2)
print(a)
```

```
# 50.aliasing, viewing and copying an array.
from numpy import *

a = arange(1, 6)  # create array a
b = a  # b is alias to a
# b = a.view() # b is shallow copy of a
# b = a.copy()  # b is deep copy of a

print('Original array: ', a)
print('Alias array: ', b)

b[0]=99  # modify 0th element of b

print('After modification: ')
print('Original array: ', a)
print('Alias array: ', b)
```

```
# 51.create and display a 2D array
import numpy
arr = numpy.array([[1,2,3], [4,5], [10,11,12]])

# display only rows
for i in range(len(arr)):
    print(arr[i])

# display all elements
for i in range(len(arr)):
    for j in range(len(arr[i])):
        print(arr[i][j], end=' ')
    print()
```

```
# 52.matrix operations
import numpy as np

a = np.matrix([[1,2],[3,4]])
b = np.matrix([[2,2],[4,4]])

c = a*b
print(c)

d = a/2
print(d)
```

```
# 53.finding transpose
import numpy
r,c = [int(a) for a in input('How many rows cols? ').split(' ')]
str = input('Enter elements:\n')

m = numpy.matrix(str)
x = numpy.reshape(m, (r,c))

print('original\n', x)
print('transpose\n', x.transpose())
```

```
# 54.finding transpose - to accept elements in matrix form
import numpy
r,c = [int(a) for a in input('How many rows cols? ').split(' ')]
```

```
arr = numpy.zeros((r,c))   # create a rXc 2D array

print('Enter elements:')
for i in range(r):   # enter one row at a time
    arr[i]= [float(x) for x in input().split(' ')]

m = numpy.matrix(arr)

print('original\n', m)
print('transpose\n', m.transpose())
```

```
# 55.nested list with names and marks
lst = ['raju', [40,50,60], 'vani', [60,70,80], 'teja', [33,44,55]]

name = input('Enter name: ')
for i in range(len(lst)):
    if lst[i]==name:
        print('Hai', name, end=' ')
        print('Your marks: ', end=' ')
        for x in lst[i+1]:
            print(x, end=' ')
        break
else:
    print('Name not found')
```

```
# 56.a list comprehension to get even nos upto 10

evens1 = [x for x in range(2, 11, 2)]
evens2 = [x for x in range(2, 11) if x%2==0]

print(evens1)
print(evens2)
```

```
# 57.adding the elements of two lists
x = [10, 20, 30]
y = [40, 50, 60]

# without using list comprehension
z = []
for i in range(len(x)):
    z.append(x[i]+y[i])
print(z)

# now using list comprehension
z = [x[i]+y[i] for i in range(len(x))]
print(z)
```

```
# 58.to retrieve common elements of two lists
x = [10, 20, 40, -1, 20]
y = [40, 50, 20, -1, 0]

# without using list comprehension
z = []
for i in x:
    if i in y:
        z.append(i)
print(z)

# now using list comprehension
z = [i for i in x if i in y]
```

```
print(z)
```

```
# 59.to retrieve only the first letter of each word in the list
words = ['Apple', 'Grapes', 'Banana', 'Orange']

# without using list comprehension
lst = []
for i in words:
    lst.append(i[0])

print(lst)

# now using list comprehension
lst = [i[0] for i in words]
print(lst)
```
