Kavayitri Bahinabai Chaudhari
North Maharashtra University, Jalgaon
T. Y. B. Sc. (Computer Science)
(w.e.f. June -2020)
DSC SEC(UG-CS-505)Python Programming – I
Semester-V
Total lectures: 45
Total Marks: 90

Course Objectives:

 The course is designed to provide Basic knowledge of Python.

 Python programming is intended for software engineers, system analysts, program managers and user support personnel who wish to learn the Python programming language.

 To learn how to design and program Python applications.

 To develop problem solving skills and their implementation through Python.

 Master the fundamentals of writing Python scripts

Course Outcome: At the end of the course, the student will be able to

 Explain basic principles of Python programming language

 Construct and apply various filters for a specific task.

 Apply the best features of mathematics, engineering and natural sciences to program real life problems.

Unit – 1 Introduction to Python Programming L: 5 M:10

 Introduction to Python

 History of Python

 Version of Python

 Need, Featuresof Python

 Applications of Python

 Installing Python on Linux and Windows

 Installing Python IDE

Unit – 2Basics of Python Programming L:10 M: 20

 Python Identifiers, Variables and Keywords

 Putting Comments

 Expressions and Statements

 Standard Data Types – Basic, None, Boolean, Numbers.

 Type Conversion Function

 Operators in Python

 Operator Precedence

 Accepting Input and Displaying Output

Flow Control Statements
 Conditional Statements

 Looping Statements

 break, continue, pass Statements

Unit – 3Python Strings L: 10 M: 20
 Introduction to String

 String Literals

 Assign String to a Variable

 Multiline Strings

 Operations on Strings, Index Operator: Working with the Characters of a String, String Methods, Length, The Slice Operator, String Comparison,

 Concepts of Python Lists: Creating, Initializing and Accessing elements in lists, Traversing, Updating and deleting elements from Lists.

 List Operations: Concatenation, List Indexing, Slices

 Built- in List functions and methods

 Aliasing, Cloning Lists

Unit – 4Python Tuples and Dictionary L: 10 M: 20
Introduction to Tuples
 Creating Tuples.

 Deleting Tuples.

 Accessing elements in a Tuple.

 Tuples Operations: Concatenation, Repetition, Membership, Iteration.

 Built- in Tuples functions and methods

Introduction to Dictionary
 Dictionaries: Concept of key-value pair.

 Creating, Initializing and Accessing elements in a Dictionary.

 Traversing, Updating and Deleting elements in a Dictionary

 Built- in Dictionary functions and methods

Unit – 5Python Functions and ModulesL: 10 M: 20
Introduction to Functions
 Defining a Function (def)

 Calling a Function

 Function Arguments - Required arguments, Keyword arguments,

Default arguments, Variable-length arguments
 Scope of Variables

 Void functions and function returning values

 Recursion

 Advance Function Topics: Anonymous Function Lambda, Mapping Functions, Functional Programming Tools: filter and reduce

Introduction to Modules

 Creating Modules and Packages

 Importing Modules

 Using the dir() Function

 Built-in Modules

Unit – 1 Introduction to Python Programming

- **Introduction to Python**
- History of Python
- Version of Python
- Need, Features of Python
- Applications of Python
- Installing Python on Linux and Windows
- Installing Python IDE

Introduction to Python:

Python is an easy to learn, powerful programming language. It has a simple but effective approach to efficient high-level data structures and object-oriented programming. Python's elegant syntax and dynamic typing, along with its defined format, make it an ideal language for scripting and rapid application development across multiple platforms.

Python is a great language with which you can learn programming. There are many reasons for this, But the simple explanation is that it is easy to read and quick to write; It doesn't take long to come to the fore Working code that does something meaningful. Python has a very human-friendly syntax, which makes Writing beautiful code is easy.

Python is a high-level general-purpose programming language:

- Because Python scripting language, web application implementation language, etc. as the code was automatically compiled and executed for byte code. So it is suitable for use.
- Because Since Python can be extended to C and C ++, Python can also be accelerated to perform awkward tasks.
- Because Due to its consistent use of structured constructs (nested code blocks, functions, classes, modules, and packages) and objects and object-oriented programming, Python enables you to write clear, logical write applications for small and large tasks.

Overview of Python:

Python is a high-level, defined, interactive, and object-oriented scripting language. Python is designed to be highly readable. English keyword is frequently used where other languages use punctuation and has less synthetic construction than other languages.

- A scripting language Python is suitable for
    - For embedding,
    - For writing a short unstructured scripts,
    - For "quick and dirty" programs.
- It is Dynamic - For example:
    - Types are bound to values, not to variables.
    - Function and method lookup is done at runtime.
    - Values are inspect able.
    - There is an interactive interpreter, more than one, in fact.
    - You can list the methods supported by any given object.

- Automatic garbage collection. (But, there is a gc module to allow explicit control of garbage collection.)
- Comparison with other languages: compiled languages (e.g. C/C++); Java; Perl, Tcl, and Ruby. Python excells at: development speed, execution speed, clarity and maintainability.
- Python is interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is interactive: Provides interactive command line and interpreter shell. In fact, there are many. You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented: Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language: Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python:

Python was developed by Guido van Rossum in the late 1980s and its implementation was started in December 1989 at the Centrum Wiskunde & Informatica (CWI) is a National Research Institute for Mathematics and Computer Science in the Netherlands. It first appeared in 1991. Python 2.0 was released in the year 2000 and Python 3.0 was released in the year 2008.

The language was named Python after the famous British television comedy show Monty Python's Flying Circus, which was one of Guido's favorite television programmes.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, UNIX shell, and other scripting languages. Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Version of Python:

It first appeared in February 1991 Van Rossum published the code (labeled version 0.9.0) to alt sources. In 1994, Python 1.0 was released with new features like: lambda, map, filter, and reduce. Python 2.0 was released in 2000 and it added new features like: list comprehensions, garbage collection system. On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language.

Python programming language is being updated regularly with new features and supports. There are lots of updations in python versions, started from 1994 to current release.
A list of python versions with its released date is given below.

Python Version	Released Date
Python 1.0	January 1994
Python 1.5	December 31, 1997
Python 1.6	September 5, 2000
Python 2.0	October 16, 2000

Python 2.1	April 17, 2001
Python 2.2	December 21, 2001
Python 2.3	July 29, 2003
Python 2.4	November 30, 2004
Python 2.5	September 19, 2006
Python 2.6	October 1, 2008
Python 2.7	July 3, 2010
Python 3.0	December 3, 2008
Python 3.1	June 27, 2009
Python 3.2	February 20, 2011
Python 3.3	September 29, 2012
Python 3.4	March 16, 2014
Python 3.5	September 13, 2015
Python 3.6	December 23, 2016
Python 3.7	June 27, 2018

Table: List of Python Versions

Why Python?

According to the Institute of Electrical Electronics and Electronics Engineers (IEEE) 201, Python ranks third after C and Java. Python Job Market Search ranks fifth, according to www.indeed.co.in dated on 2016. Clearly, all the data shows the growing demand in the Python job market. This language is great if you just want to learn for fun or build your career around Python. Python has a major role to play in surprisingly new technologies. Whether it's a cloud platform, mobile app development, BigData, IoT with Raspberry Pi or new block chain technology, Python is being seen as a niche language platform for developing and delivering scalable and robust deliver applications.

Popular Interpreters of Python

By definition, an interpreter is a type of computer program that directly executes the instructions written in some programming or scripting language. By direct execution, we mean that it doesn't require the instructions to be compiled into a machine language program first.

1) CPython

Supports – Up to Python 3.7

It is the default and most widely-used implementation of the Python programming language. Written in C and Python, CPython is an interpreter that offers a foreign function interface with C and other programming languages.

CPython can also be classified as a compiler because it converts the Python code into bytecode before interpreting the same.

As the reference implementation of Python, CPython offers the most compatibility with Python packages and C extension modules. As such, all versions of the Python programming language are implemented in C.

CPython is the only option for using Python packages that rely on C extensions to function properly. Targeting CPython is a must if one wishes to reach the widest possible audience for a program developed in the Python programming language.

2) *IronPython*

Supports – Up to Python 2.7

An implementation of the Python programming language for the .NET framework, IronPython uses both Python and .NET framework libraries. It can also expose Python code to other languages pertaining to the .NET framework. IronPython provides support for dynamic compilation and comes with an interactive console. Using the Python interpreter allows Python scripts to interact with .NET objects. With the Python Tools for Visual Studio, IronPython comes directly integrated into the Visual Studio IDE.

3) *Jython*

Supports – Up to Python 2.7

Formerly known as JPython, Jython is an implementation of Python that runs on the Java platform. Written in Java and Python, Jython converts Python code into Java bytecode and hence allows Python code to run on any machine having a JVM.

Jython provides support for both static and dynamic compilation. An important feature of the popular Python interpreter is that it allows importing as well as using any Java class like a Python module.

Whether you need to interface with an existing Java codebase or write Python code for the JVM, you can place your bet on Jython.

4) *PyPy*

Supports – Up to Python 2.7, Python 3.5 and Python 3.6

PyPy is a fast, compliant alternative implementation of the Python programming language. The Python interpreter is implemented in RPython, which is a restricted statically-typed subset of the Python programming language.

Featuring a JIT compiler, PyPy supports C, CLI, and JVM back-ends. The foremost goal of PyPy is to offer maximum compatibility with the reference CPython implementation while enhancing the performance at the same time.

Need of Python

Python is one of the top 10 popular programming languages of 2017. Python is a general purpose and high level programming language. You can use Python for developing desktop GUI applications, websites and web applications. Also, Python, as a high level programming language, allows you to focus on core functionality of the application by taking care of common programming tasks. The simple syntax rules of the programming language further makes it easier for you to keep the code base readable and application maintainable. There are also a number of reasons why you should prefer Python to other programming languages.

6 Reasons Why You Must Consider Writing Software Applications in Python

1) Readable and Maintainable Code

While writing a software application, you must focus on the quality of its source code to simplify maintenance and updates. The syntax rules of Python allow you to express concepts without writing additional code. At the same time, Python, unlike other programming languages, emphasizes on code readability, and allows you to use English keywords instead of punctuations. Hence, you can use Python to build custom applications without writing additional code. The readable and clean code base will help you to maintain and update the software without putting extra time and effort.

2) Multiple Programming Paradigms

Like other modern programming languages, Python also supports several programming paradigm. It supports object oriented and structured programming fully. Also, its language features support various concepts in functional and aspect-oriented programming. At the same time, Python also features a dynamic type system and automatic memory management. The programming paradigms and language features help you to use Python for developing large and complex software applications.

3) Compatible with Major Platforms and Systems

At present, Python is supports many operating systems. You can even use Python interpreters to run the code on specific platforms and tools. Also, Python is an interpreted programming language. It allows you to you to run the same code on multiple platforms without recompilation. Hence, you are not required to recompile the code after making any alteration. You can run the modified application code without recompiling and check the impact of changes made to the code immediately. The feature makes it easier for you to make changes to the code without increasing developmenttime.

4) **Robust Standard Library**

Its large and robust standard library makes Python score over other programming languages. The standard library allows you to choose from a wide range of modules according to your precise needs. Each module further enables you to add functionality to the Python application without writing additional code. For instance, while writing a web application in Python, you can use specific modules to implement web services, perform string operations, manage operating system interface or work with internet protocols. You can even gather information about various modules by browsing through the Python Standard Library documentation.

5) ***Many Open Source Frameworks and Tools***

As an open source programming language, Python helps you to curtail software development cost significantly. You can even use several open source Python frameworks, libraries and development tools to curtail development time without increasing development cost. You even have option to choose from a wide range of open source Python frameworks and development tools according to your precise needs. For instance, you can simplify and speedup web application development by using robust Python web frameworks like Django, Flask, Pyramid, Bottle and Cherrypy. Likewise, you can accelerate desktop GUI application development using **Python GUI frameworks** and toolkits like PyQT, PyJs, PyGUI, Kivy, PyGTK and WxPython.

6) Simplify Complex Software Development

Python is a general purpose programming language. Hence, you can use the programming language for developing both desktop and web applications. Also, you can use Python for developing complex scientific and numeric applications. Python is designed with features to facilitate data analysis and visualization. You can take advantage of the data analysis features of Python to create custom big data solutions without putting extra time and effort. At the same time, the data visualization libraries and APIs provided by Python help you to visualize and present data in a more appealing and effective way. Many **Python developers** even use Python to accomplish artificial intelligence (AI) and natural language processing tasks.

Features of Python:

Some key features of the python are:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintain.
- Python programs can run on any platform, you can carry code created in Windows machine and run it on Mac or Linux.
- Python has inbuilt large library with prebuilt and portable functionality, also known as the standard library.
- Python is an expressive language.
- Python is free and open source.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- Python can be both dynamically and strongly typed, dynamically typed means it is a type of variable that is interpreted at runtime, which means, in Python, there is no need to define the type (int or float) of the variable.
- Python is Scalable, Interpreted and Interactive language.
- It supports automatic garbage collection.
- Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles.
- Python provides interfaces to all major commercial databases.
- Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix

Applications of Python:

Python is known for its general-purpose nature that applies it to every domain of software development. The dragon can be used in any area of development.

One of the most famous platforms where Python is extensively used is YouTube. The other places where you will find Python being extensively used are the special effects in Hollywood movies, drug evolution and discovery, traffic control systems, ERP systems, cloud hosting, e-commerce platform, CRM systems, and whatever field you can think of.

Here, we are specifying the application area where the Python can be applied.

1. **Web Applications:** We can use Python to develop web applications. HTML & XML, JSON, Email Processing, Request, Bundlesup, Feed Parser etc. Libraries are available to handle similar Internet protocols. It can be used to design and deliplop web based applications like Django, Pyramid, Flask etc. Also provides a similar framework. There are some important developments: PythonWikiIngence, Poku, PythonBlogSoftware etc.
2. **Desktop GUI Applications:** Python provides Tk GUI library to develop user interface in python based application. Some other useful toolkits wxWidgets, Kivy, pyqt that are useable on several platforms. The Kivy is popular for writing multitouch applications.
3. **Software Development:** Python is helpful for software development process. It works as a support language and can be used for build control and management, testing etc.
4. **Scientific and Numeric:** Python is popular and widely used in scientific and numeric computing. Some useful library and package are SciPy, Pandas, IPython etc. SciPy is group of packages of engineering, science and mathematics.
5. **Business Applications:** Python is used to build Bussiness applications like ERP and e-commerce systems. Tryton is a high level application platform.
6. **Console Based Application:** We can use Python to develop console based applications. For example: IPython.
7. **Audio or Video based Applications:** Python is awesome to perform multiple tasks and can be used to develop multimedia applications. Some of real applications are: TimPlayer, cplay etc.
8. **3D CAD Applications:** To create CAD application Fandango is a real application which provides full features of CAD.
9. **Enterprise Applications:** Python can be used to create applications which can be used within an Enterprise or an Organization. Some real time applications are: OpenErp, Tryton, Picalo etc.
10. **Applications for Images:** Using Python several applications can be developed for image. Applications developed are: VPython, Gogh, imgSeek etc.

Installing Python on Linux and Windows

Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

Installing Python on UNIX and Linux:

Here are the simple steps to install Python on Unix/Linux machine.

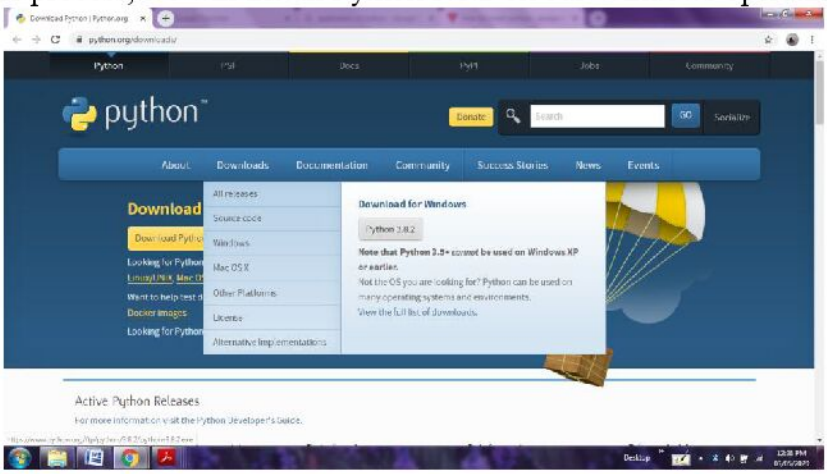
- Open a Web browser and go to **http://www.python.org/download/.**
- Follow the link to download zipped source code available for Unix/Linux.
- Download and extract files.
- Editing the Modules/Setup file if you want to customize some options.
- **run ./configure** script
- make
- make install

This installs Python at standard location /usr/local/bin and its libraries at /usr/local/lib/pythonXX where XX is the version of Python.

Installing Python on Windows:

Here are the simple steps to install Python on Windows machine.

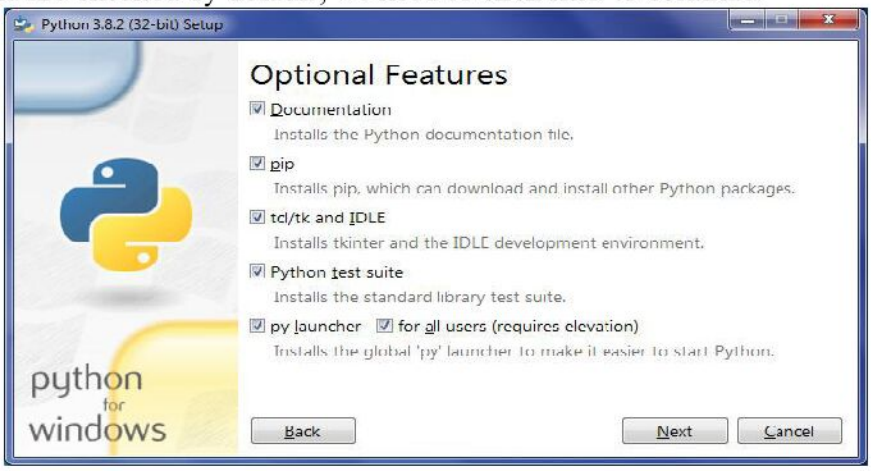
- Visit the link **https://www.python.org/downloads/** to download the latest release of Python. In this process, we will install Python 3.8.2 on our Windows operating system.



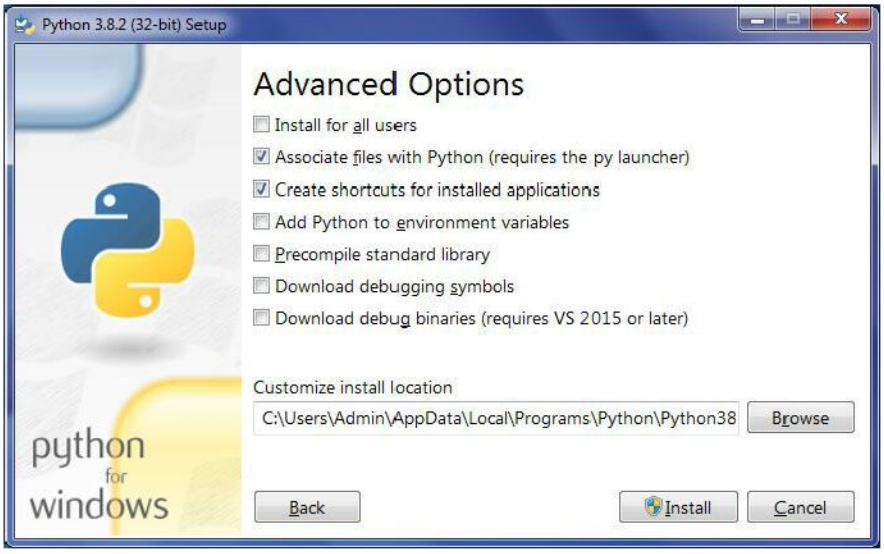
- Double-click the executable file which is downloaded; the following window will open. Select Customize installation and proceed.



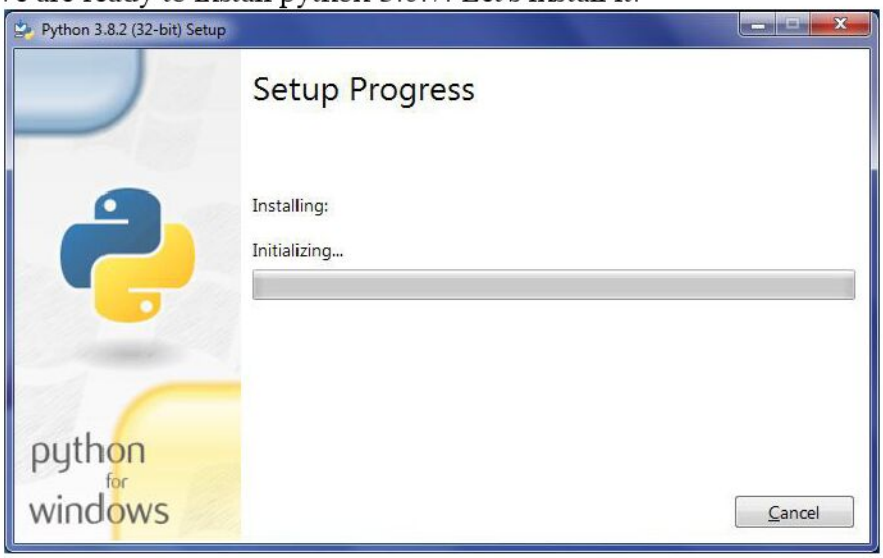
- The following window shows all the optional features. All the features need to be installed and are checked by default; we need to click next to continue.



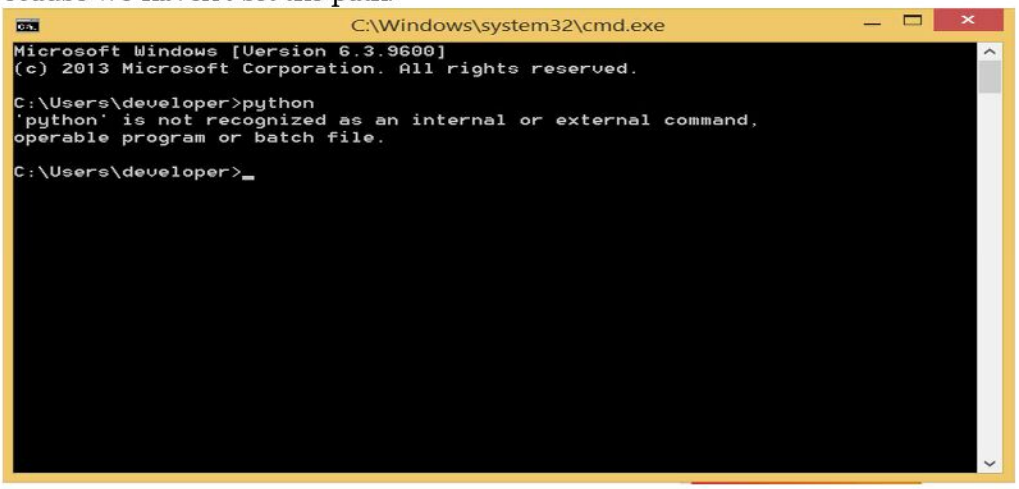
- The following window shows a list of advanced options. Check all the options which you want to install and click next. Here, we must notice that the first check-box (install for all users) must be checked.



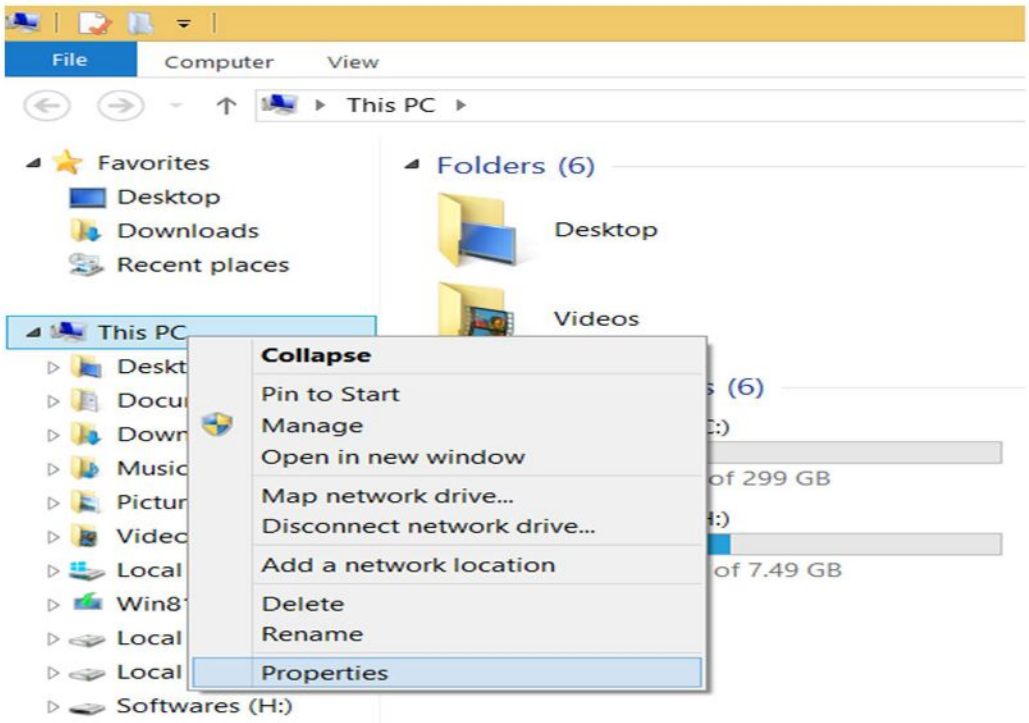
- Now, we are ready to install python-3.6.7. Let's install it.

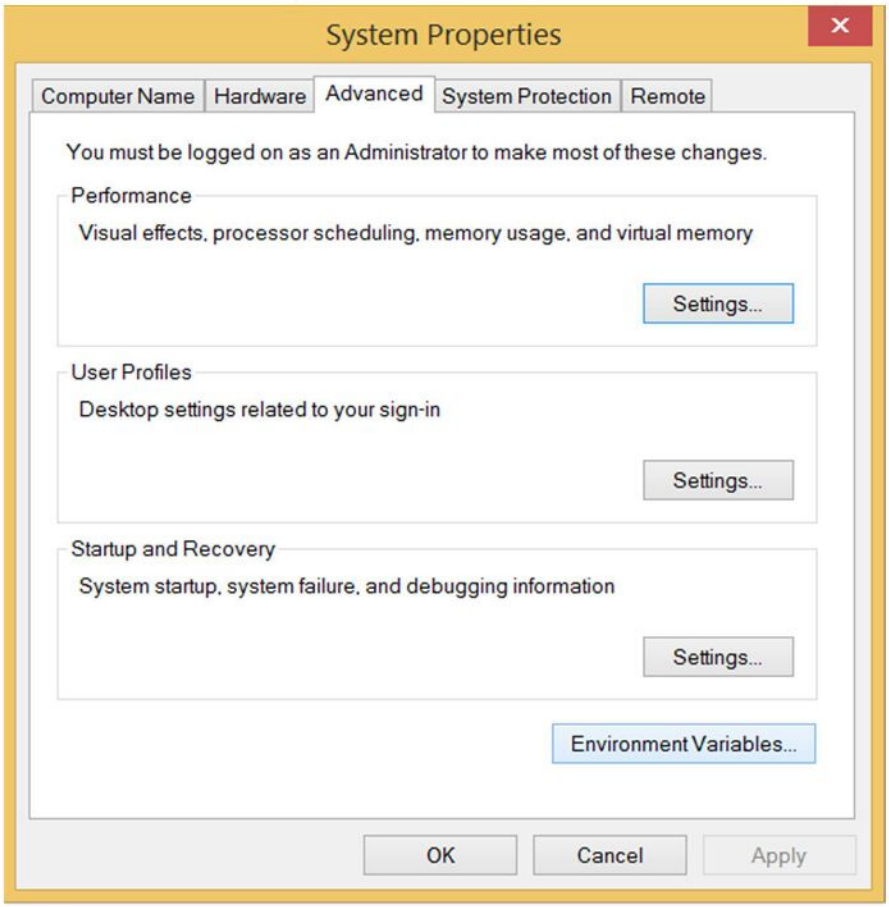


- Now, try to run python on the command prompt. Type the command python in case of python2 or python3 in case of python3. It will show an error as given in the below image. It is because we haven't set the path.

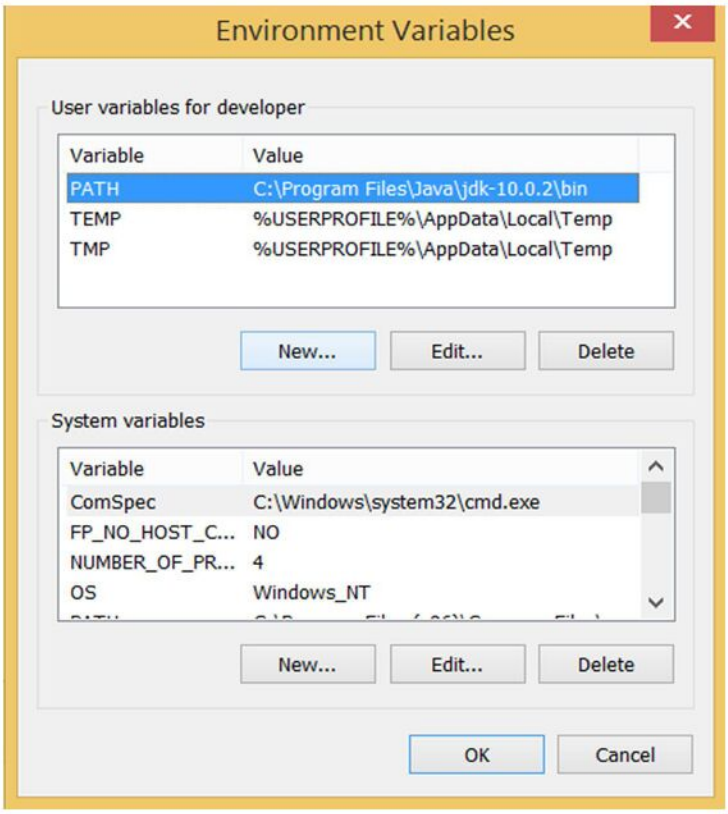


- To set the path of python, we need to the right click on "my computer" and go to Properties → Advanced → Environment Variables.

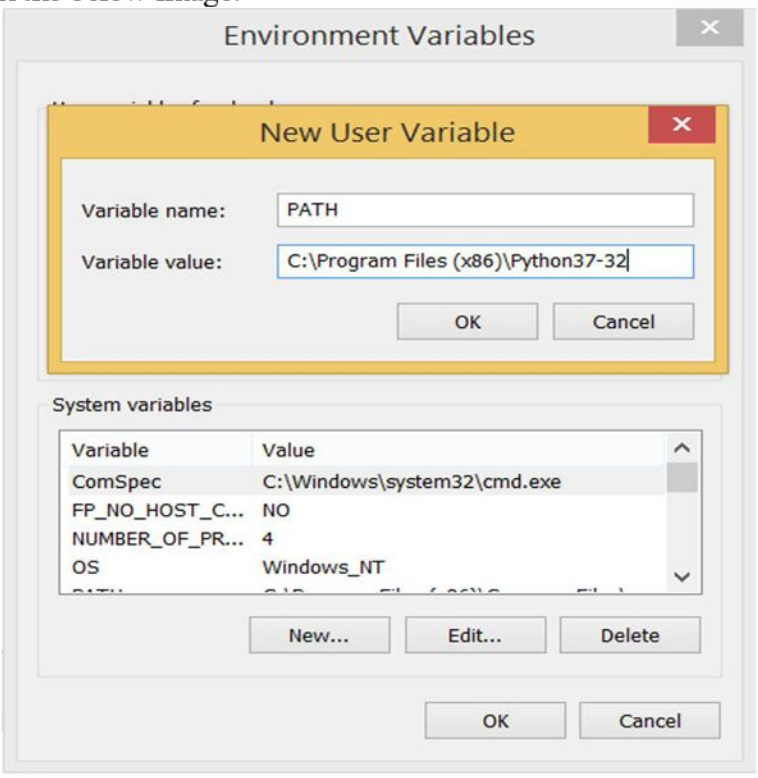

File
Computer
View
This PC
Favorites
Desktop
Downloads
Recent places
This PC
Folders (6)
Desktop
Videos
Collapse
Pin to Start
Manage
Open in new window
Map network drive...
Disconnect network drive...
Add a network location
Delete
Rename
Properties
Softwares (H:)


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

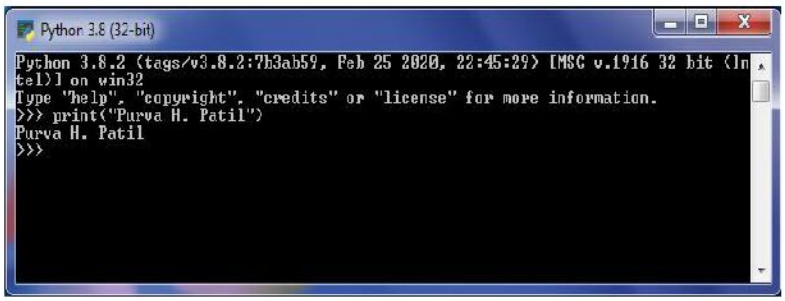
- Add the new path variable in the user variable section.



- Type PATH as the variable name and set the path to the installation directory of the python shown in the below image.



- Now, the path is set, we are ready to run python on our local system. Restart CMD, and type python again. It will open the python interpreter shell where we can execute the python statements.
- **Example**
  - Open the python interpreter shell
  - Type - print("Purva H. Patil")



  - You get Output – Purva H. Patil

Python IDE:

Code and debug programs are easily created for Python code editor developers. Using this Python IDE (Integrated Development Environment) you can manage a large codebase and deploy quickly. Developers can use these editors to create desktop or web applications.
Many Python IDEs are available. The most useful Python IDE are as Follows:

1. **PyCharm:** PayCharm is a cross-platform IDE used for Python programming. This editor can be used on Windows, macOS, and Linux. This software contains API that can be used by the developers to write their own Python plugins so that they can extend the basic functionalities.
   Features of PyCharm
   - It is an intelligent Python code editor supports for CoffeeScript, JavaScript, CSS, and TypeScript.
   - Provides smart search to jump to any file, symbol, or class.
   - Smart Code Navigation
   - It offers quick and safe refactoring of code.
   - It allows you to access PostgreSQL, Oracle, MySQL, SQL Server, and many other databases from the IDE.

2. **Spyder:** Spyder is a scientific integrated development environment written in Python. This software is designed for and by scientists who can integrate with Matplotlib, SciPy, NumPy, Pandas, Cython, IPython, SymPy, and other open-source software. Spyder is available through Anaconda (open-source distribution system) distribution on Windows, macOS, and Linux.
   Features of Spyder
   - Allows you to run Python code by cell, line, or file.
   - Plot a histogram or time-series, make changes in dateframe or numpy array.
   - It offers automatic code completion and horizontal/vertical splitting.
   - Find and eliminate bottlenecks

- An interactive way to trace each step of Python code execution.

3. **IDLE:** IDLE (Integrated Development and Learning Environment) is a default editor that comes with Python. This software helps a beginner to learn Python easily. IDLE software package is optional for many Linux distributions. The tool can be used on Windows, macOS, and Unix.

   Features of IDLE
   - Search multiple files
   - It has an interactive interpreter with colorizing of input, output, and error messages.
   - Supports smart indent, undo, call tips, and auto-completion.
   - Enable you to search and replace within any window.

4. **Sublime Text 3:** Sublime Text 3 is a code editor which supports many languages including Python. It has basic built-in support for Python. Customization of Sublime Text 3 is available for creating create a full-fledged Python programming environment. The editor supports OS X, Windows, and Linux operating systems.

   Features of Sublime Text 3
   - Allows you to highlight syntax.
   - It has command Palette implementation that accepts text input from users.
   - Handle UTF8 BOMs in .gitignore files
   - Display badges for folders and file to indicate Git status
   - Changes to a file are represented by markers available in the gutter.

5. **Visual Studio Code:** Visual Studio Code (VS Code) is an open-source environment developed by Microsoft. This IDE can be used for Python development. Visual Studio Code is based on Electron which is a framework to deploy Node JS applications for the computer running on the Blink browser engine.

   Features of Visual Studio Code
   - The editor provides smart code completion based on function definition, imported modules, as well as variable types.
   - You can work with Git as well as other SCM providers
   - Enable you to debug code from the editor.
   - Provides extensions to add new languages, debuggers, themes to gain the advantage of additional services.

6. **Atom:** Atom is a useful code editor tool preferred by programmers due to its simple interface compared to the other editors. Atom users can submit packages and them for the software.

   Features of Atom
   - Package Manager Integrated for Plugins support
   - The feature of smart auto completion
   - Supports Command Palette
   - Multiple panes
   - Allow cross-platform editing

7. **Jupyter:** Jupyter is a tool for people who have just started with data science. It is easy to use, interactive data science IDE across many programming languages that just not work as an editor, but also as an educational tool or presentation.
   Features of Jupyter
   - It supports for Numerical simulation, data cleaning machine learning data visualization, and statistical modeling.
   - Combine code, text, and images.
   - Support for many programming languages.
   - Integrated data science libraries (matplotlib, NumPy, Pandas).

8. **Pydev:** PyDev is a third-party Python editor for Eclipse. This editor can be used in not only Python but IronPython and Jython development.
   Features of Pydev:
   - It has interactive console shortcuts
   - Allows you to create a Google App Engine (GAE) Python project
   - Find and Go to definition
   - Automatically import code to complete it.
   - You can Configure Django integration.

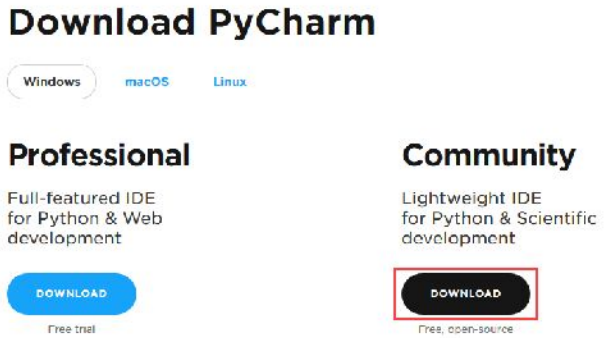
9. **Thonny:** Thonny is an IDE for learning and teaching programming, specially designed with the beginner Pythonista scripting environment. It is developed at The University of Tartu, which you can download for free on the Bitbucket repository for Windows, Linux, and Mac.
   Features of Thonny:
   - Allows developers to view how their code and shell commands affect Python variables.
   - It has a simple debugger.
   - Provides support for evaluating an expression.
   - Python function call opens a new window with separate local variables table as well as code pointer.
   - Automatically spot syntax error.

10. **Wing:** Wing is a lightweight Python environment which is designed to give you productive development experience.
    Features of Wing
    - Immediate feedback by writing your Python code.
    - Helps you to remove common errors and write better Python code.
    - You can check for debug data and try out bug fixes interactively without restarting your app.
    - Wing supports test-driven development with various frameworks like the unittest, pytest, nose, doctest, and Django testing.

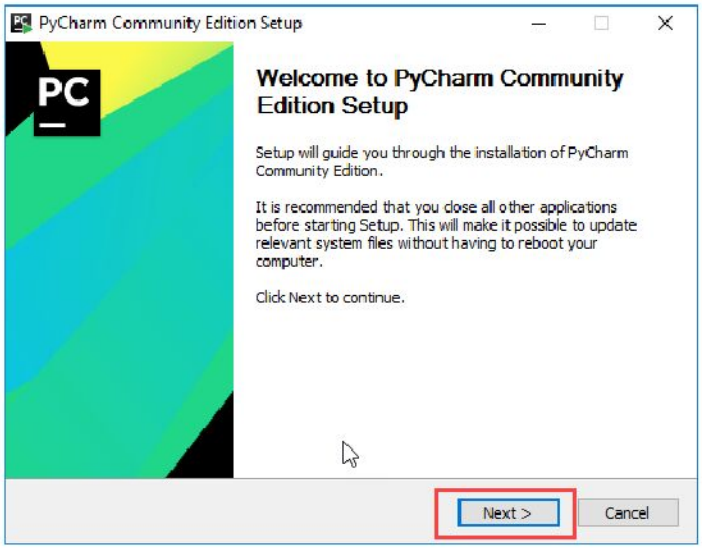
Installing Python IDE (Installing Pycharm):

The steps of Installing Python IDE – Pycharm are as follows

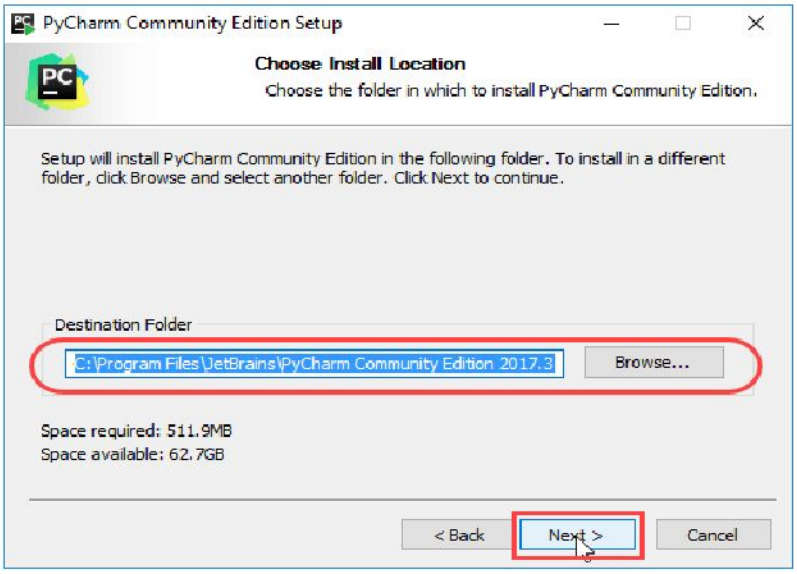
1. To download PyCharm on https://www.jetbrains.com/pycharm/download/ and Click the "DOWNLOAD" link under the Community Section.



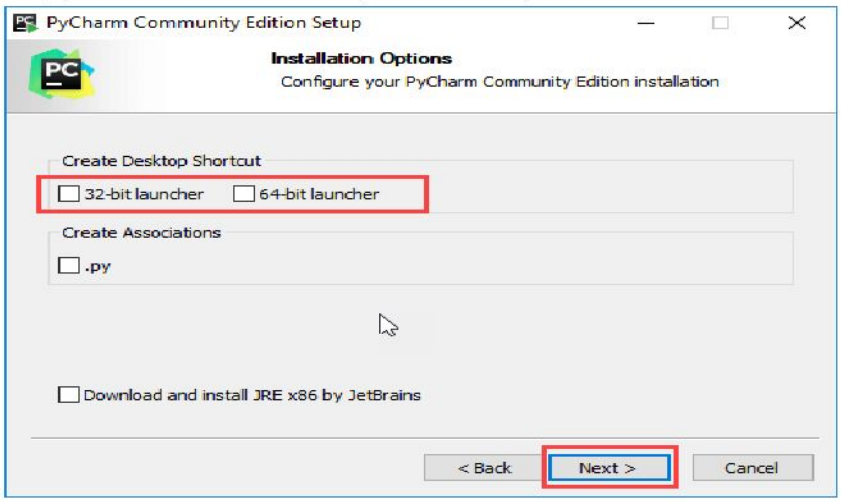
2. Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click "Next".



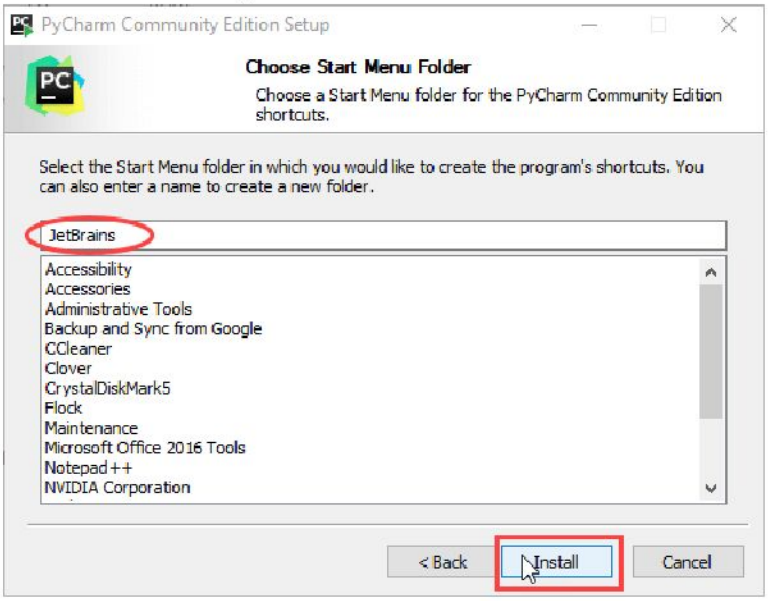
3. On the next screen, Change the installation path if required. Click "Next".



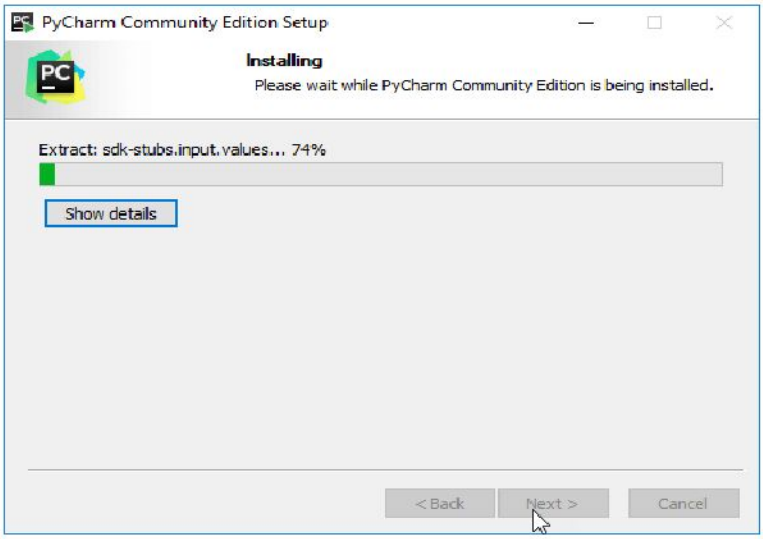
4. On the next screen, you can create a desktop shortcut if you want and click on “Next”.



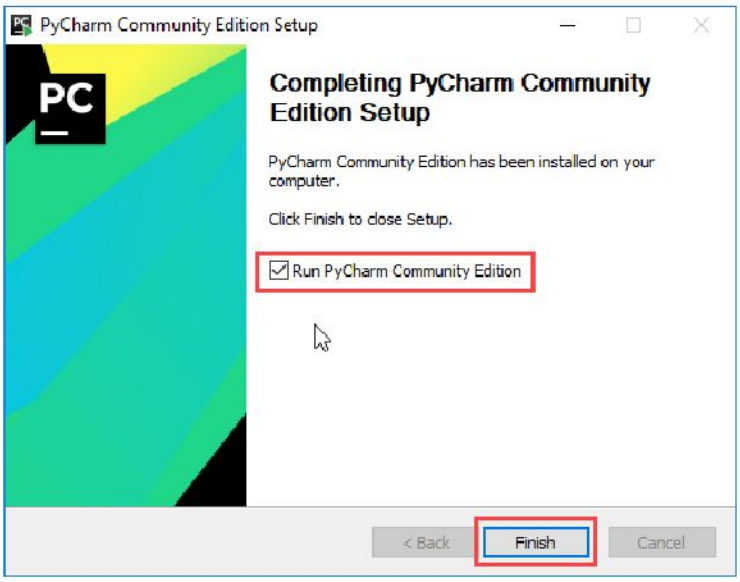
5. Choose the start menu folder. Keep selected JetBrains and click on “Install”.



6. Wait for the installation to finish.



7. Once installation finished, you should receive a message screen that PyCharm is installed



8. After you click on "Finish," the Following screen will appear.



Unit – 2 Basics of Python Programming

Python Identifiers, Variables, and Keywords

Python Identifiers

- An identifier in Python is a name used for identifying a variable, class, function, object, module.
- An identifier in python programming starts with a letter A to Z / a to z / an underscore (_) followed by zero or more letters, underscores, and digits.
- Python language does not allow punctuation characters within an identifier name for example: @, $, and %.
- No identifiers can begin with a number, and no symbols other than the underscore.
- Python is a case sensitive programming language. Hence, **Python_Identifier** and **python_identifier** are two different identifiers in Python.

Python variables

- Variables are containers for storing a value.
- Python is dynamically typed, it means that there is no requirement of pre-declaration of a variable or its type. The type (and value) are initialized on the assignment. Assignments are performed using an equal sign (=).
- Following example depicts how to declare variables in python:

```
>>> count = 0

>>> name = 'Ram'

>>> count = count + 1
```

- A variable is created the moment you first assign a value to it. Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 4            # x is of type int
x = "Tally"      # x is now of type str
print(x)
```

- String variables in Python can be declared either by using single or double quotes:

```
x = "GDB"
# is the same as
x = 'GDB'
```

A variable can have a short name like (a, b) or a more descriptive name like (age, bookname, total_volume). Rules for Python variables are the same as for identifiers. The rules for an identifier are given above. The following table shows some legal and illegal variable names.

#Legal variable names	#Illegal variable names
myvar = "Gargi" my_var = "Gargi" _my_var = "Gargi" myVar = "Gargi" MYVAR = "Gargi" myvar2 = "Gargi"	2myvar = "HBP" my-var = "HBP" my var = "HBP"

Python keywords

- Keywords in python are the reserved words.
- These words cannot be used as a variable name, function name, or any other identifier name.
- The following table lists some keywords of python.

and	**as**	**assert**	**Break**
class	**continue**	**Def**	**Del**
elif	**else**	**except**	**Exec**
finally	**for**	**from**	**Global**
if	**import**	**In**	**Is**
lambda	**not**	**Or**	**Pass**
print	**raise**	**return**	**Try**
while	**with**	**yield**	**None**

Putting Comments

- Comments are an important part of the program because specifying comments in your code helps you and others to understand later on the intention of the code.
- They can be used to make the code more readable.
- They can also be used to prevent execution when testing code.

- Single line comment can be specified by using the # symbol in python.
  For example :

```
#This is a comment
print("Hello")
```

- Multiline comment:
  Python does not have a syntax for multi-line comments. If you wish to add a multiline comment you could insert a # for each line, as like in following code:

```
#This is a comment
#written in
#more than just one line
print("Hello !")
```

or, another way is that you can use a multiline string. Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it, for instance:

```
"""
This is a comment
written in
more than just one line
"""
print("Hello !")
```

- **Expressions and Statements**

Expressions

Expression in any programming language is a combination of values, variables, and operators. A value is also considered expression and same with a variable, so the following are all legal expressions (assume that the variable x has been assigned with some value):

```
x

x + 17
```

If you type an expression in interactive mode, the interpreter **evaluates** it and displays the result:

```
>>> 2 + 2

4
```

Statements

A statement is a part of code that is executed by the Python interpreter. When you type a statement in interactive mode, the interpreter executes it and displays the result, if there is one. A script usually contains a sequence of statements. If there is more than one statement, the results appear one at a time as the statements execute.

For example, the script

```
print(5)

x = 3

print(x)
```

produces the output

```
5
```

3 and the assignment statement produces no output.

Standard Data Types – Basic, None, Boolean, Numbers.

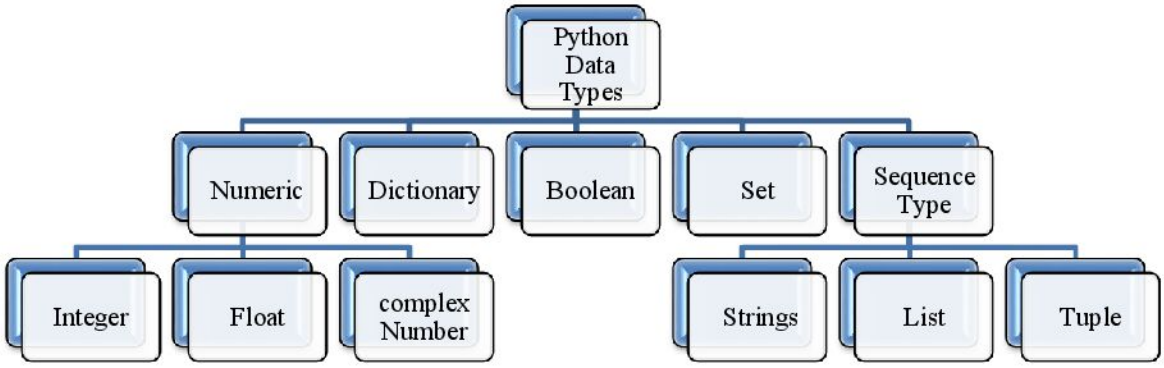


Fig: Some data types available in Python

Many different types of data can be stored in memory. For example, an employee's salary is stored as a numeric value, his/her address is stored as alphanumeric characters. Python provides different standard data types that are used to define the operations possible on them and the storage method for each of them. We are going to discuss some of them here.

Numeric

Numeric data type represents the data that has a numeric value in python. The numeric value can be an integer, floating-point number, or even a complex number. These values are defined as int, float, and complex class in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal).
- **Float** – This value is represented by float class. It is a real number with floating-point representation. It is specified by a decimal point.
- **Complex Numbers** – Complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

Note – type() function is used to determine the type of data.

```
# Python program to demonstrate the numeric value

a = 5

print("Type of a: ", type(a))
```

```
b = 5.0

print("\n Type of b: ", type(b))


c = 2 + 4j

print("\n Type of c: ", type(c))
```

Output:

```
Type of a:  <class 'int'>

Type of b:  <class 'float'>

Type of c:  <class 'complex'>
```

Boolean

Data type with one of the two built-in values, True or False. True and False with a capital 'T' and 'F' are valid booleans otherwise python will throw an error.

```
# Python program to demonstrate the boolean type

  print(type(True))

print(type(False))
```

Output:

```
<class 'bool'>

<class 'bool'>
```

None

None data type means nothing, not any value. For instance: not at all, not a bit, etc. The **None Data type** is used to define the **null value** or **no value**, the none value means not 0, or False value, and it is a data it's own.

Example:

```
a=None
```

None is also internally represented as an object, consider a=None, that means a is officially not pointing to any value; instead, it is pointing to none. If it is an object, and if you can type as a print(id(a)) you will get an address. To know the type of you can execute print(type(a)) then it will reply as 'NoneType'.

```
a=None
print(id(a))
print(type(a))
```

Inside Python, **only one None object can be created,** it means that, if you assign None to many variables, all the variables are going to represent the same object.

Sequence Type

A sequence is an ordered collection of similar or different data types. Python has the following built-in sequence data types:

- **String**: A string value is a collection of one or more characters put in single, double, or triple quotes.
- **List**: A list object is an ordered collection of one or more data items, not necessarily of the same type, put in square brackets.
- **Tuple**: A Tuple object is an ordered collection of one or more data items, not necessarily of the same type, put in parentheses.

Dictionary

A dictionary object is an unordered collection of data in a key:value pair form. A collection of such pairs is enclosed in curly brackets. For example: {1:"Steve", 2:"Bill", 3:"Ram", 4: "Farha"}

Type Conversion Function

Sometimes, it may be needed to perform conversions between the built-in types. To convert between types, you can simply use the type name as a function. There are a number of built-in functions to perform the conversion from one data type to another. These functions return a new object that represents the converted value. Some of them are given in the following table:

Function & its Description

int(a [,base]) : Converts a to an integer. Here base specifies the base if a is a string.
long(a [,base]): Converts a to a long integer where base specifies the base if a is a string.
float(x) : Converts x to a floating-point number.
complex(real [,imag]): Creates a complex number.
str(a) : Converts object a to a string representation.
tuple(a) : Converts a to a tuple.
list(a) : Converts a to a list.
set(a) : Converts a to a set.
dict(d) : Creates a dictionary. d must be a sequence of (key,value) tuples.
chr(a) : Converts an integer to a character.
hex(a) : Converts an integer to a hexadecimal string.
oct(a) : Converts an integer to an octal string.

Operators in Python

- Operators are used to perform operations on variables and values. For example consider the expression 4 + 5 = 9 where 4 and 5 are operands and + is an operator.
- Python language supports the following types of operators.
  1. **Arithmetic Operators**
  2. **Comparison (Relational) Operators**
  3. **Assignment Operators**
  4. **Logical Operators**
  5. **Bitwise Operators**
  6. **Membership Operators**
  7. **Identity Operators**

Following subsections details each type of operators along with some examples.

1. **Arithmetic Operators**

Operator	Description	Example Assume variable a and b holds some value. Execute on shell and observe the result.
+	Performs additions of both operands.	a + b
-	Performs subtractions, subtract right hand operand from left hand operand.	a – b
*	Performs multiplication of both operands.	a * b
/	Performs division, divides left hand operand by right hand operand	b / a
%	Performs division of left hand operand by right hand operand and returns remainder	b % a
**	Performs exponential (power) calculation on operators	a**b
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 = 4

2. Comparison (Relational) Operators

Comparison operators compare the values of both side operands and decide the relation exists among these operands. Hence they are also called as relational operators.

Operator	Description	Example
==	This operator checks if the values of both operands are equal, if yes then returns true.	a == b
!=	This operator checks if the values of both operands are not equal, if values are not equal then returns true.	a != b
<>	This operator checks if the values of both operands are not equal, if values are not equal then returns true.	a <> b
>	This operator checks, if the value of left hand operand is greater than the value of right hand operand, if yes the condition is true.	a > b
<	This operator checks, if the value of left hand operand is less than the value of right hand operand, if yes, then condition is true.	a < b

>=	This operator checks if the value of left hand operand is greater than or equal to the value of right hand operand, if yes then the condition is true.	a >= b
<=	This operator checks if the value of left hand operand is less than or equal to the value of right hand operand, then condition becomes true.	a <= b

3. **Assignment Operators**

Operator	Description	Example
=	Assigns values from right hand operands to left hand operand	c = a + b
+=	It adds right hand operand to the left hand operand and assign the result to left hand operand	c += a
-=	It subtracts right hand operand from the left operand and assign the result to left hand operand.	c -= a
*=	It multiplies right hand operand with the left hand operand and assign the result to left hand operand	c *= a
/=	It divides left operand with the right operand and assign the result to left operand	c /= a
%=	It takes modulus using two operands and assign the result to left operand	c %= a
**=	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a
//=	It performs floor division on operators and assign value to the left operand	c //= a

4. **Logical Operators**

Operator	Description	Example
and	This operator returns true if both the operands are true.	a and b
or	This operator returns true if any of the two operands are non-zero.	a or b
not	It reverses the logical state of its operand.	Not a

5. **Bitwise Operators**

Operator	Description	Example
&	This operator copies a bit to the result if it exists in both operands	a & b
|	It copies a bit if it exists in any one operand.	a | b
^	It copies the bit if it is set in one operand but not set in both.	a ^ b
~	It is unary and has the effect of 'flipping' bits.	~a
<<	The left hand operands value is moved to left by the number of bits specified by the right hand operand.	a << 2
>>	The left hand operands value is moved to right by the number of bits specified by the right hand operand.	a >> 2

6. **Membership Operators**

Python has membership operators which are not available in many programming languages. These membership operators are use to test for membership in a sequence, like strings, lists, or tuples. There are two membership operators as explained in following table.

Operator	Description	Example
in	This operator returns true if it finds a variable in the specified sequence otherwise it returns false.	x in y
not in	This operator returns true if it does not find a variable in the specified sequence otherwise it returns false.	x not in y

7. **Identity Operators**
   Identity operators are use to compare the memory locations of the two objects. There are two identity operators exists in python as explained in below section.

Operator	Description	Example
is	This operator assess true if the variables on either side of the operator point to the same object otherwise it returns false.	x is y
is not	This operator assess to false if the variables on either side of the operator point to the same object otherwise it returns true.	x is not y

Operator Precedence

The following table lists all operators from highest precedence to lowest.

Sr.No.	Operator & Description
1	**
2	~ + -
3	* / % //
4	+ -
5	>><<
6	&
7	^ |
8	<= <>>=
9	<> == !=
10	= %= /= //= -= += *= **=
11	is is not
12	in not in
13	not or and

Accepting Input and Displaying Output

- Python provides two inbuilt functions to read the input from the keyboard.
    1. **raw_input** ()
    2. **input** ()

1. raw_input ( ) : This function works in older version. It takes exactly what is typed from the keyboard, convert it into string and then return it to the variable in which we want to store it. For example –

```
# Python program showing a use of raw_input()

a = raw_input("Enter your name : ")
```

2. input ( ) : This function initially takes the input from the user and then evaluates the expression, it means Python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python. For example –

```
# Python program showing a use of input()

v = input("Enter your value: ")

print(v)
```

Display output

- Python uses print() function to display output. For example

```
v="abc"
print(v)
```

Flow Control Statements

Conditional Statements

Conditional Statement in Python performs different computations or actions depending on whether a specific Boolean constraint evaluates to true or false. Conditional statements are handled by if statements in Python.

In Python, If Statement is used for decision making. It will run the body of code only when condition mentioned in if statement is true.

Syntax

```
if expression:
                Statement
else:
      Statement
```

Some examples demonstrating the use of if statement and its variations:

1) Example 1

```
a = 3
b = 20
if b > a:
   print("b is greater than a")
```

2) Example 2

```
a = 33
b = 33
if b > a:
     print("b is greater than a")
elif a == b:
      print("a and b are equal")
```

3) Example 3

```
a = 200
b = 33
if b > a:
   print("b is greater than a")
elif a == b:
   print("a and b are equal")
else:
   print("a is greater than b")
```

4) Example 4

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

Looping Statements

Python programming language provides several types of loops as given below to handle looping requirements.

While loop

1) *Syntax*

```
While test_exp:

    Body of loop
```

- The while loop first checks the test expression (test_exp). The body of the loop is entered only if the test_exp evaluates to true. After each iteration, the test expression is checked again. This process continues until the test_exp evaluates to false.
- In Python, no {} are their to determine the body of the while loop instead it is determined through indentation.
- The body of loop starts with indentation and the first unindented line denotes the end of the loop.

 # Program to add natural numbers up to 15

 # sum = 1+2+3+...+15

```
num = 15

# initialize sum and counter
sum = 0
```

```
cnt = 1


while cnt <= num:
    sum = sum + cnt
    cnt = cnt+1    # update counter


# print the sum
print("The sum is", sum)
```

For loop

- A for loop in python is generally used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- For example:

```
for x in "GDB":
        print(x)
```

break, continue, pass Statements

Using **for loops** and while loops in Python allow you to automate and repeat tasks in an efficient manner. But sometimes, an external factor may influence the way your program runs. When this occurs, you may want your program to exit a loop completely, skip part of a loop before continuing, or ignore that external factor. You can do these actions with continue, break and pass statements.

Break Statement

In Python programming language, the break statement provides you the chance to exit out of a loop when an external condition is generated. You can place the break statement within the block of code below the loop statement, typically after if statement.

Let's look at an example that uses the break statement in a for loop:

```
number =0

for number inrange(5):
```

```
if number ==3:
break# break here

print('Number is '+str(number))

print('Out of loop')
```

When we run this code, we get following output:
Number is 0
Number is 1
Number is 2
Out of loop

This shows that once the integer number is evaluated as equivalent to 3, the loop breaks, The break statement causes a program to break out of a loop.

Continue Statement

- The continue statement gives you the option to skip over the part of a loop where an external condition is triggered, but to go on to complete the rest of the loop..
- The continue statement will be within the block of code under the loop statement, usually after a conditional if statement.
- Using the same for loop program as mentioned above we'll use a continue statement rather than a break statement:

```
number =0

for number inrange(5):
if number ==3:
continue# continue here

print('Number is '+str(number))

print('Out of loop')

Output
Number is 0
Number is 1
Number is 2
Number is 4
Out of loop
```

Here, Number is 3 never occurs in the output, but the loop continues after that point to print lines for the numbers 4 before leaving the loop. The continue statement causes a program to skip certain factors that come up within a loop, but then continue through the rest of the loop.

Pass Statement

- When an external condition is triggered, the pass statement allows you to handle the condition without the loop being impacted in any way; all of the code will continue to be read unless a break or other statement occurs.
- As with the other statements, the pass statement will be within the block of code under the loop statement, typically after a conditional if statement.
- Using the same code block as above, let's replace the break or continue statement with a pass statement:

```
number =0

for number inrange(5):
if number ==2:
pass# pass here

print('Number is '+str(number))

print('Out of loop')


Output
Number is 0
Number is 1
Number is 2
Number is 3
Number is 4
Out of loop
```

By using the pass statement in this program, we notice that the program runs exactly as it would if there were no conditional statement in the program. The pass statement tells the program to ignore that condition and continue to run the program as usual.

MCQs Unit II

1. Which of the following character is not allowed while declaring Python identifier?
    a. $
    b. A
    c. 5
    d. _
2. Which of the following is correct variable declaration in Python?
    a. my_var = "ABC"
    b. 2myvar = "ABC "
    c. my-var = "ABC "
    d. my var = "ABC "
3. Which of the following is not valid Python keyword?
    a. assert
    b. lambda
    c. finally
    d. printf
4. Which symbol is use to specify single line comment in Python?
    a. $
    b. #
    c. ‘
    d. “
5. Which is not valid data type in Python?
    a. Integer
    b. Strings
    c. List
    d. Paragraph
6. What is output of following Python statement?

    print(type(True))

    a. <class 'Int'>
    b. <class 'complex'>
    c. <class 'bool'>
    d. <class 'boolean'>
7. Which of the following is not valid sequence type in Python?
    a. List
    b. Tuple
    c. String
    d. Integer
8. Key:Value pair is present in which data type?
    a. List
    b. Tuple
    c. String
    d. Dictionary

9. Which of the following operator denotes floor division in Python?
    a. //
    b. /
    c. *
    d. **
10. Which of the following is not correct bitwise operator?
    a. &
    b. |
    c. ~
    d. ∨
11. Which of the following is membership operator?
    a. not in
    b. not is
    c. is
    d. not
12. Which of the following is an identity operator?
    a. not in
    b. not is
    c. is
    d. not
13. Which of the following function is use to accept input in Python?
    a. Raw_num()
    b. raw_ip()
    c. Raw_txt()
    d. raw_input ()
14. Which of the following statement is not used within the loop?
    a. break
    b. continue
    c. pass
    d. discontinue
15. Which of the following Python code is correct?
    a.
```
for x in "GDB":
        print(x)
```
    b.
```
for x :
        print(x)
```
    c.
```
for "GDB":
        print(x)
```
    d.
```
for x in ([GDB]):
        print(x)
```
16. Which of the following is use to determine the body of the while loop?
    a. indentation
    b. {}
    c. ()

d. []

17. Which of the following statement is use in Python to display the output?
    a. printf()
    b. print()
    c. print_txt()
    d. println_Text()
18. Which of the following is valid in Python?
    a. if.....else
    b. if...elif....else
    c. both a & b
    d. none
19. which of the following operator has highest precedence in Python?
    a. **
    b. *
    c. //
    d. /
20. ..... operator has the effect of 'flipping' bits.
    a. <<
    b. ^
    c. ~
    d. >>

Question	Details
Q.1	Define Python Identifier
Q.1	Define Python variable
Q.1	Define expression
Q.1	Define statement
Q.1	How to specify single line and multiline comments in Python?
Q.1	Write the syntax of if statement in Python.
Q.1	Write the syntax of while statement in Python.
Q.1	Write the syntax of for statement in Python.
Q.1	What is use of break statement?
Q.1	What is use of pass statement?
Q.1	What is use of continue statement?
Q.1	Which function is use to read input from user in python? Write the syntax of it.
Q.2	What is meant by python identifier? Give some examples.
Q.2	What is meant by a variable? Give some example that demonstrates legal and illegal variables in python.
Q.2	What is mean by keywords in Python. Give any four examples of valid keywords in

	python programming.
Q.2	What are uses of providing comments in program?
Q.2	Explain numeric data type in Python.
Q.2	What is use of type() function in Python? Explain with example.
Q.2	Explain any four arithmetic operators available in Python with help of example.
Q.2	Explain identity operators available in Python with help of example.
Q.2	Explain membership operators available in Python with help of example.
Q.2	Explain logical operators available in Python with help of example.
Q.2	With help of suitable example explain if..else statement in Python.
Q.2	With help of suitable example explain while loop statement in Python.
Q.3	Write the rules for declaring identifiers in Python.
Q.3	Explain any two sequence type in Python.
Q.3	Explain any three type conversion functions with suitable example.
Q.3	Explain arithmetic operators in Python.
Q.3	Explain Comparison Operators in Python.
Q.3	Explain assignment operators in Python.
Q.3	Explain bitwise operators in Python.
Q.3	Explain operator precedence in Python.
Q.3	With the help of suitable example explain for loop in Python.
Q.3	With suitable code explain how to accept input and display output in Python.
Q.3	Explain if…elif..else statement in Python with example.
Q.3	What is use of pass, continue and break statement in python.
Q.4	What is a variable? What are the rules for defining a variable in Python? Explain with help of suitable examples.
Q.4	What are uses of comments? How to specify single and multiline comments in Python? Explain with suitable code.
Q.4	Explain numeric data type in Python with the help of suitable example.
Q.4	Explain with help of example sequence type

	in python. Which function is use to find the data type of a variable in python.
Q.4	Explain with proper syntax any six functions use for type conversions in Python.
Q.4	Explain with suitable example all arithmetic operators available in Python.
Q.4	Explain all comparison operators available in Python with help of suitable examples.
Q.4	Explain all assignment operators available in Python with help of suitable examples.
Q.4	Explain membership and identity operators available in Python with help of suitable examples.
Q.4	Explain all variations of if statement available in Python with help of suitable examples.
Q.4	Explain with example while and for loop in Python.
Q.4	How to read integer, string and floating point number as input in Python? Explain with suitable example. Also explain how to display output in Python.
Q.5	Write a short note on following: A) Identifiers B) Variables
Q.5	Write a short note on standard data types available in Python.
Q.5	Write a short note on following: A) Logical operators B) Bitwise operators
Q.5	Explain in detail the operator precedence in python.
Q.5	Write a short note on conditional statements in Python.
Q.5	Write a short note on looping statements in Python.
Q.5	Write a short note on following statements in Python: A) Break B) Continue C) Pass
Q.5	Write a Python program that displays the multiplication table of given number. [Note: Read the number from user as input.]
Q.5	Write a python program that calculates simple and compound interest.
Q.5	Write a python program that displays factorial

	of a given number.
Q.5	Write the python program that checks given number is prime or not prime.
Q.5	Write a program in Python that adds natural numbers up to 15

Unit – 3 Python Strings

- **Introduction to String**
  - A string is a data type that is used to represent text rather than numbers and is comprised of a set of characters that can also contain spaces and numbers. For example, the word "python" and the phrase "Python programming" are both strings.
  - We can create them simply by putting the characters in quotes.
  - Python treats single quotes the same as double-quotes.
  - Creating strings is very simple and it is same as assigning a value to a variable.
  - Example

    s1='Gargi'

    s2="Python Programming"

- **String Literals**

  - In python programming language, string literals are enclosed by either single quotation marks, or double quotation marks. For example the two string literals 'hello' , "hello" are same. You can use print() function to display a string literal.

  - **Example**

    print("Python Programming")

- **Assign String to a Variable**

  In Python programming we can assign a string value to a variable through the variable name followed by an equal sign. The following syntax shows how to assign string to a variable.

  s = "welcome"

  we can display the value in string variable by using print() function.

  print(s)

- **Multiline Strings**
  - In python a **multiline string** can be specified by using either three single quotes or three double quotes at the beginning and end of the multiline string.

- Any quotes, tabs, or newlines in between the “triple quotes” are considered part of the **string**.
- **Python's** indentation rules for blocks do not apply to lines inside a **multiline string**.
- **Example**

```
>>> a = """Python is a good programming language
It is an Object Oriented Programming Language."""
>>> print(a)
```

- **Operations on Strings**

We can perform various operators like substring, concatenation, updating string value, appending string, etc on string.

Substring

To access substrings from a given string in python we can use the square brackets for slicing along with the index/indices for obtaining desired substring.

For example –

```
>>> s='hello'

>>> print(s[1:4])
```

Output :

ell

Updating Strings

To update an existing string we can reassign a variable with another string. The new updated value can be related to its previous value or it can be a completely different string. For example:

```
>>> v="Hello World"
>>> v=v[0:5]+"Python"
>>> print(v)
```

When the above code statements are executed, the following result will be produced:

Output : HelloPython

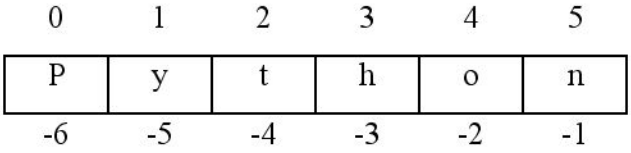
String Special Operators

Assume string variable **a** holds 'Hello' and variable **b** holds 'Python', then –

Operator	Description	Example
+	Concatenation - Adds values on both side of the operator	a + b will gives HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will gives – HelloHello
[]	Slice - Gives the character from the given index	a[1] will gives e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will gives ell
in	Membership - Returns true if a character exists in the given string	H in a will gives 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will gives 1

Index Operator: Working with the Characters of a String,

The **index operator** selects a single character from a string. In Python square bracket is use to include the index. The characters are accessed by their position or index value. For example, in the string shown below, the 6 characters are indexed left to right from position 0 to position 5.



It is also the case that the positions are named from right to left using negative numbers where -1 is the rightmost index and so on.

```
>>> s="Python"
>>> s[2]
```

Will produce following output

't'

```
>>> s[-1]
```

Will produce following output

'n'

The expression in brackets is called an **index**. The index *indicates* which character you want. It can be any integer expression so long as it evaluates to a valid index value.

String Methods

To find the length of given string Python uses len() method. The following example demonstrates the use of length method.

>>> s="Python Programming"

>>> len(s)

Output: 18

Following table summarizes some important methods that can be used with strings.

Method	**Description**	**Example** **s='abc'**
capitalize()	Converts the first character to upper case	s.capitalize()
count()	Returns the number of times a specified value occurs in a string	s.count('a')
endswith()	Returns true if the string ends with the specified value	s.endswith('c')
find()	Searches the string for a specified value and returns the position of where it was found	s.find('b')
format()	Formats specified values in a string	
index()	Searches the string for a specified value and returns the position of where it was found	s.index('b')
isalnum()	Returns True if all characters in the string are alphanumeric	s.isalnum()
isalpha()	Returns True if all characters in the string are in the alphabet	s.isalpha()
isdecimal()	Returns True if all characters in the string are decimals	s.isdecimal()
isdigit()	Returns True if all characters in the string are digits	s.isdigit()
isidentifier()	Returns True if the string is an identifier	s.isidentifier()

islower()	Returns True if all characters in the string are lower case	s.islower()
isnumeric()	Returns True if all characters in the string are numeric	s.isnumeric()
isupper()	Returns True if all characters in the string are upper case	s.isupper()
lower()	Converts a string into lower case	s.lower()
replace()	Returns a string where replacement is performed	s.replace('b','p')
split()	Splits the string at the specified separator, and returns a list	s.split()
splitlines()	Splits the string at line breaks and returns a list	s.splitlines()
startswith()	Returns true if the string starts with the specified value	s.startswith('a')
strip()	Returns a trimmed version of the string	s.strip()
upper()	Converts a string into upper case	s.upper()

The Slice Operator,

In Python language we can use slice operator with strings. Slice operator is use to access any substring within given range. The slice operator with a single argument will give us a single character, and the slice operator with a range, i.e., using a colon (:), will give us multiple consecutive characters. Following example depicts the use of slice operator.

>>> a="Python Programming"

>>> a[0]

Output : 'P'

>>> a[2:5]

Output : 'tho'

>>> a[17]

Output : 'g'

>>> a[:10]

Output : 'Python Pro'

>>> a[5:]

Output : 'n Programming'

String Comparison

We can use comparision operator to compare strings following code demonstrates some example that how comparison is carried out. Strings are compared lexicographically (ASCII value order) When the value comparison operators are used.

>>> a='abc'

>>> b='pqr'

>>> c='xyz'

>>> a < b

Output : True

>>> a == b

Output : False

>>> a != c

Output : True

>>> a < c and b =='xyz'

Output : False

- **Concepts of Python Lists**

Sequence is the most basic data structure available in python. Every element of a sequence has assigned with a number, which denotes its position or index. Always The first index is zero, the second index is one, and so on. Python has six built-in types of sequences, but the most common are lists and tuples.

Lists and tuples can be thought of as generic "arrays" with which to hold an arbitrary number of arbitrary Python objects. The items are ordered and accessed via index offsets, similar to arrays, except that lists and tuples can store different types of objects. Here in this section we are going to see lists in detail.

Creating list:

>>> new_list=[]

>>> new_list

Output: []

>>> list1=[1,2,3,4,5]

>>> list1

Output: [1, 2, 3, 4, 5]

>>> list2=['abc','def','hij']

>>> list2

Output: ['abc', 'def', 'hij']

>>> list3=[1, 'abc', 100.0]

>>> list3

Output: [1, 'abc', 100.0]

Initializing and Accessing elements in lists:

>>> new_list=[]

>>> new_list=[1,2,3]

>>> new_list

Output: [1, 2, 3]

Accessing Values in Lists

Square brackets [] along with index/indices are used for accessing values in the lists. For example :

>>> list3[1]

Output: 'abc'

>>> list3[0:2]

Output: [1, 'abc']

Traversing list:

>>> char_list=['a','b','c','d']

>>> for x in char_list: print(x)

Output:

a

b

c

d

Updating list:

Updating Lists

The lists can be updated for single or multiple elements. append() method can also be use to update the list. If you want to update the list you can use the slice operator along with assignment operator. Refer the following statements for updating a list.

>>> new_list=[1,2,3]

>>> new_list[1]="abc"

>>> new_list

Output: [1, 'abc', 3]

Deleting elements from Lists:

Delete List Elements

We can use del statement to remove a list element, if we know exactly which element(s) to be deleted or otherwise the remove() method can be used. del statement takes the index while the remove() takes the actual value of the element. For example −

>>> new_list=[1, 'abc', 3]

>>> del new_list[1]

>>> new_list

Output: [1, 3]

>>> new_list.remove(3)

>>> new_list

Output: [1]

- **List Operations: Concatenation, List Indexing, Slices**

We can concatenate two lists using + operator. For example following expression demonstrates list concatenations:

>>> [1,2,3]+['a','b','c']

Output: [1, 2, 3, 'a', 'b', 'c']

>>> lista=[11,12,13]

>>> listb=[21,22,23]

>>> lista+listb

Output: [11, 12, 13, 21, 22, 23]

List indexing

We can access individual element in the list by specifying its index. The valid list index is any number between 0 to (length of that list-1). List indexing and slicing is similar as string indexing which we had already discussed in previous section. Following expression demonstrates list indexing.

>>> lista=[1,2,3,'a','b',100.05,205.4]

>>> lista

Output: [1, 2, 3, 'a', 'b', 100.05, 205.4]

>>> lista[3]

Output: 'a'

>>> lista[-3]

Output: 'b'

List slicing

We can access more than one element in sequence from the list by taking the slices of the list. Since lists are sequences, indexing and slicing work the same way for lists as they work for strings. Following examples demonstrates how to take list slices.

>>> lista[0:]

Output: [1, 2, 3, 'a', 'b', 100.05, 205.4]

>>> lista[2:5]

Output: [3, 'a', 'b']

>>> lista[:5]

Output: [1, 2, 3, 'a', 'b']

- **Built- in List functions and methods**

Following table presents some of list functions provided in python –

Sr.No.	Function with Description

1	cmp(list1, list2): this function compares elements of both lists.
2	len(list): it providedes the total length of the list.
3	max(list): it returns the element in list who has maximum value.
4	min(list): it returns the element in list who has minimum value..
5	list(seq): this function converts a tuple into a list.

Python includes following list methods

Sr.No.	Methods with Description
1	list.append(obj): it appends (insert at last position) object obj to list
2	list.count(obj) : it returns count that how many times obj occurs in list
3	list.index(obj): it returns the first index in list that obj appears
4	list.insert(index, obj): it inserts object obj into list at an index
5	list.remove(obj): it removes object obj from list
6	list.reverse(): it reverses objects of list in place
7	list.sort([func]): it Sorts objects of list, use compare func if given

- **Aliasing, Cloning Lists**

Aliasing a List

- Aliasing a list means creating another reference for the same list.
- When we change the original list the changes are reflected in the aliased list also.
- Only one list is created with two different references. [Note: you can verify it using id()].

```
List1=[10,20,30,40,50]
>>> List2=List1      #creates alias for List1.
>>> List1
[10, 20, 30, 40, 50]
>>> List2
[10, 20, 30, 40, 50]
>>> id(List1)    # id for list1
```

56531984

>>> id(List2) # id for list2

56531984

- Ids for both List1, List2 are same means both lists refer to the same list. We can verify the same things by appending an element in any one list. The following code depicts that if we append an element in one list the changes are reflected in the another list also.

  >>> List1.append(55)

  >>> List1

  [10, 20, 30, 40, 50, 55]

  >>> List2

  [10, 20, 30, 40, 50, 55]

Cloning a list

Cloning is the process of creating the exactly same copy from the original copy. When we create the alias of a list, then if we change one list the changes are reflected in alias list also. If we don't want this thing to be happened then we clone the list. The following code depicts how to create clone of the list.

```
List1=[11,12,13]
>>> List2=List1[:]   #clone of List1
>>> List1
[11, 12, 13]
>>> List2
[11, 12, 13]
>>> id(List1)
56532664
>>> id(List2)
56532064
```

- Ids of both List1, List2 are different means there are two different lists denoting to List1 and List2. Now we will append the element in List1 and verify that does List2 contains it or not.

```
>>> List1.append(14)
>>> List1
[11, 12, 13, 14]
>>> List2
[11, 12, 13]
```

- List1.append(14) only adds 14 in List1 not in List2. Hence List1 and List2 both refer to two different lists.

MCQs Unit III

1. Which of the following operator is use to concatenate two strings in Python?
   a. +
   b. *
   c. /
   d. ++
2. Which of the following denotes range slice in Python?
   a. [:]
   b. []
   c. :
   d. (:)
3. To find the length of given string, Python uses …….. method.
   a. Size()
   b. length()
   c. len()
   d. SizeOf()
4. Which of the following function returns the number of times a specified value occurs in a Python string?
   a. count()

b. size()

c. time()

d. return()

5. Which of the following denotes rightmost index of a string s in Python?
   a. s[-1]

   b. s[l]

   c. s[1]

   d. s[n]

6. ..... function compares elements of both lists.
   a. compare(list1, list2)

   b. comparision(list1, list2)

   c.check(list1, list2)

   d.cmp(list1, list2)

7. Which of the following statement is use to clone the Python list?
   a.List2=list.list[1]

   b.List2.list1

   c. List2=List1[:]

   d.List2=List1

8. Which of the following function provides the total length of the list?
   a. len(list)

   b. length(list)

   c. count(list)

   d. size(list)

9. We can access individual element in the list by specifying its.....
   a. value

   b. index

   c. size

d. ASCII value

10. We can use ……statement to remove a list element.
    a. delta

    b. del

    c. insert

    d. index

11. Which of the following are used for accessing values in the lists?
    a.square brackets []

    b. index/indices

    c.square brackets [] along with index/indices

    d. none

12. Python strings are compared …… .
    a.lexicographically

    b. graphically

    c. both a & b

    d. none

13. ……..operator is use to access any substring within given range.
    a. Range

    b.Slice

    c. Repetition

    d. concatenation

14. The …….. operator selects a single character from a string.
    a. indent

    b. identity

    c.index

    d. membership

15. Which of the following returns a trimmed version of the string?
    a. trim()

b. del()

c. remove()

d.strip()

16. Which of the following denotes repetition operator for strings?

a.*

b. +

c. []

d. [:]

17. How to denote multiline strings in Python?

a. enclosing string in """....."""

b. enclosing string in ###.....###

c. enclosing string in <<<.....>>>

d.enclosing string in $$$.....$$$

18. In python programming language, string literals are enclosed by .....marks.

a. single quotation
b. double quotation
c. both a & b
d. none

19. Which of the following is correct to display the value in string variable s?

a.print(s)

b. s

c. printf(S)

d. "S"

20. Which of the following is valid in Python?

a. s=='welcome"

b. s.welcome

c. s-"welcome"

d.s = "welcome"

Question	Details
Q.1	Describe string data type in Python.
Q.1	How to specify multiline string in Python?
Q.1	How to take substring from a string in Python?
Q.1	How to concatenate python strings?
Q.1	Describe * operators used with strings in Python.
Q.1	Give the syntax and use of count() method used with python strings.
Q.1	What is python list?
Q.1	How to create and initialize a list in python?
Q.1	How to access elements in list?
Q.1	How to update an already existing list in Python?
Q.1	How to traverse a list of characters in Python?
Q.1	How to delete an element from a list?
Q.2	How to concatenate two lists in Python? Explain with help of example.
Q.2	Explain any two built in list functions in Python.
Q.2	Explain any two built in lists methods in Python.
Q.2	How to get substring and update a string in Python?
Q.2	Explain any three operators used with strings in Python?
Q.2	Explain any three methods that work with strings with help of suitable example.
Q.2	Explain Index Operator that works with the Characters of a String with

	help of suitable example.
Q.2	How to traverses a list of characters? Explain with example.
Q.2	How to delete list element and lists in Python?
Q.2	Explain list indexing with help of suitable example.
Q.2	Explain list slicing with help of suitable example.
Q.2	How to create list aliases explain with help of suitable example.
Q.3	Explain how to create and update strings with suitable code.
Q.3	Explain with suitable example + and * operators used with Python strings.
Q.3	Explain with suitable example [] and [:] operators used with Python strings.
Q.3	Explain with help of suitable example index operators for strings.
Q.3	Explain any four methods used with strings in Python.
Q.3	Explain any four methods used with lists in Python.
Q.3	How to compare two strings? Explain with help of suitable example.
Q.3	Explain how to Initialize and accessing elements in lists.
Q.3	How to traverse lists in Python? Explain with suitable example.
Q.3	How to clone lists in Python? Explain with suitable example.
Q.3	How to delete element and lists in Python? Explain with suitable example.
Q.3	Explain slice operator used in Python with help of suitable example.
Q.4	Explain with suitable code how to create single and multiline Strings in Python.
Q.4	Explain with suitable code how to create, initialize and update lists in Python.
Q.4	Explain index operator in Python with help of suitable example.
Q.4	Explain any six methods used with strings.

Q.4	Explain any six methods used with Lists.
Q.4	Explain how to create clone of a list in Python? Also explain how does it differs from aliasing?
Q.4	Explain with suitable example list indexing and slicing.
Q.4	Explain how to create list alias. How does it differs with cloning?
Q.4	Explain with example any six list operations in Python.
Q.4	Explain how to initialize and update Python Lists with help of suitable example.
Q.4	Explain string special operators in Python with help of suitable example.
Q.4	Explain with suitable example string literals in python.
Q.5	Write a short note on Strings in Python.
Q.5	Write a short note on Lists in Python.
Q.5	Write a short note on index operator in Python.
Q.5	Write a short note on methods used with strings.
Q.5	Write a short note on methods used with Lists.
Q.5	Write a short note on cloning a list. How does it differs from aliasing?
Q.5	Write a short note on list indexing and slicing.
Q.5	Write a short note on list aliasing. How does it differs with cloning?
Q.5	Write a short note on list operations in Python.
Q.5	Write a short note on initializing and updating Python Lists.
Q.5	Write a short note on string special operators.
Q.5	Write a short note on string literals in python.

Unit 4. Python Tuples and Dictionary **Marks :20**

Introduction to Tuples : A tuple is an immutable sequence of python objects. Tuple is a collection of elements similar to lists. Tuple is a collection which is ordered and unchangeable. Tuple uses round parantheses. The difference between lists and tuples is that tuple use parantheses whereas, lists uses square brackets, elements of tuple can't be changed but list can be changed.

Tuples are quite similar to list, but there are certain advantages of implements tuple over a list.

Major advantages are as follows :

Generally tuples are used for heterogeneous(different) data types while list are used for homogeneous(similar) data types.

As tuples are not change, iterating through a tuple is faster than with list. So performance of execution of tuple is slight faster than lists.

Tuples that contain immutable elements can be used as a key for a dictionary. With lists, this is not possible.

If user have a data that doesn't change and if they are implemented as a tuple then the data have guarantee of write –protected.

Creating a tuple : To create a tuple different values separated by comma are inserted in between paranthesis. This values may be of different data types like string , integer or float.

e.g. tuple1 = ("Sameer", 156, 57.8)

tuple2 = (1,2,3,4,5)

tuple3 = ("apple", "watermelon", "banana")

If we want to display contents of tuple3, we give the command

print(tuple3)

The output is : ('apple','watermelon','banana')

We can also create a tuple from unpacked elements i.e without any braces just elements separated by comma i.e.

e.g. tup1 = "Seema", 23 , "Dhule", 49.33

print(tup1)

Output : ('Seema',23,'Dhule',49.33)

The empty tuple is written as two parentheses containing nothing

e.g. tuple4 = ( )

When your tuple contain a single value then it can be written as

e.g. tup1 = (100,)

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Deleting Tuples : Tuples are unchangeable , so you cannot remove items from tuple completely. It is not possible to remove individual tuple elements but you can delete the tuple completely by using del command:

e.g. tuple1 contain name of person with its class and marks of 3 subjects

```
tuple1 = ("Sameer","TYBSc",76,78,85)
print(tuple1)
('Sameer', 'TYBSC', 76,78,85)
del tuple1
print(tuple1)
```

If we delete a tuple and then print the tuple , it gives the error message as

```
NameError: name 'tuple1' is not defined
```

Accessing elements in a Tuple : Elements of tuple can be access by referring to the index number, inside square brackets.

e.g. thistuple = ("Sameer", 156, 57.8)

```
print(thistuple[0])
```

When above code is executed it produces the following result

```
Sameer
```

If we want to print the contents of tuple at index 1, it should be as follows

```
print(thistuple[1])
156
```

We can also access the elements using negative indexing. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

e.g. thistuple = ("Sameer", 156, 57.8)

```
print(thistuple[-1])
```

When we execute the above code, it displays the last element of the tuple, which is at -1 index.

```
57.8
print(thistuple[-2])
```

This displays the second last element of tuple.

156

It is possible to access specific range of elements from tuples. To obtain a range of indexes we have specify where to start and where to end the range. When specifying a range, the return value will be a new tuple with the specified items.

e.g. tuple1 = ("Sameer", "TYBSc", 76,78,85,79.66)

print(tuple1[2:5])

When we execute the above code, it prints the element in given tuple at position 2 ,3 and 4. First item is at position 0. Searching is start at index 2 and end at index 5 but item in position 5 is not included. So the result displayed as follows

76,78,85

Tuples Operations : There are some operations perform on tuple as follows :

Concatenation: Tuples can be concatenated by using '+' operator, the result is new tuple and not a string.

e.g. tuple1 = (1,2,3)

tuple2 = ("x","y","z")

tuple3 = tuple1 + tuple2

print(tuple3)

It concatenates the tuple1 and tuple2 and produces the result as new tuple

(1,2,3,,'x','y','z')

Repetition : Sometime we need to replicate the data in a tuple. The multiplication operator is used for repetition to make multiple copies of a tuple and joins them all together.

tup1 =(1,2,3)

re_tupl = (tup1,) * 2

print(re_tup1)

It replicates the contents of tuple tupl1 and prints the result as a new tuple

(1,2,3,1,2,3)

Membership : It is used to determine if a specified item is present in a tuple. Membership in a tuple is checked using the 'in' keyword:

```
thistuple = ("apple", “watermelon”, "banana")
if  "apple" in thistuple:
  print("Yes, apple is in the fruits tuple")
```

In above code tuple thistuple contain name of fruits. Command ‘if’ checks whether apple is member of thistuple. If apple is in thistuple, it prints the give message. After executing the code it gives result : ‘Yes, apple is in the fruits tuple

e.g.
```
if 3 in (1,2,3)
  print(“Element present”)
```

After execution of above code the result is : Element present

Iteration : We can iterate over tuples using a simple for loop. Tuples are **sequence data type** and we can iterate over sequences.

e.g.
```
for x in (1, 2, 3):
  print x
```

The result is : 1,2,3

e.g.
```
fruit_names = ("apple", “watermelon”, "banana")
for fruit in fruit_names
  print(fruit)
```

The result is : ‘apple’, ‘watermelon’‘banana’

Built-in Tuples functions : Following are built in tuple functions :

len( ) : len( ) is a buit-in tuple function to compute number of items present in given.

e.g.
```
fruit_names = ("apple", “watermelon”, "banana")
print(len(fruit_names))
```

The result is : 3

cmp(tuple1,tuple2) : cmp( ) is a built-in tuple function in which elements of tuple1 are compared with elements of tuple2 , if both lists are same it returns 0 otherwise it returns 1.

e.g.
```
tuple1 = (‘x’,y’,’z’)
tuple2 = (1,2,3)
tuple3 = (‘x’,y’,’z’)
cmp(tuple1,tuple2)
output : 1
cmp(tuple1,tuple3)
```

output : 0

```
# Python program to check if a number is
# odd or even using cmp function

# check 12
n = 12
if cmp(0, n % 2):
        print"odd"
else:
        print"even"

# check 13
n = 13
if cmp(0, n % 2):
        print"odd"
else:
        print"even"
```

any() : This built-in tuple function returns true if any element present in a tuple and returns false if tuple is empty.

e.g. tuple1 = ('x','y','z')

print("Is there any element is present in tuple :", any(tuple1))

Output : Is there any element is present in tuple : True

e.g. tuple2 = ( )

print("Is there any element is present in tuple :", any(tuple2))

Output : Is there any element is present in tuple : False

min( ) : This tuple function returns smallest element(int) of the tuple

e.g. tuple1=(4,5.7,2.3,5)

print("Smallest element in tuple is :", min(tuple1))

It prints the result as : Smallest element in tuple is : 2.3

max( ) : This tuple function returns largest element(int) of the tuple

e.g. tuple1=(4,5.7,2.3,5)

print(“Largest element in tuple is :”, max(tuple1))

It prints the result as : Largest element in tuple is : 5.7

sorted( ) : It is used to sort all elements of the tuple.

e.g. tuple1=(4,5.7,2.3,5)

print(“Sorted elements in tuple are :”, sorted(tuple1))

It prints the result as : Sorted element in the tuple is : (2.3,4,5,5.7)

e.g. tuple1 = (‘e’, ‘a’, ‘u’, ‘o’, ‘i’)

print(“Sorted characters are:”, sorted(tuple1))

When we execute the code output is : Sorted characters are : (‘a’,’e’,’i’,’o’,’u’)

sum( ) : It returns sum of all elements of tuple.

e.g. Num=(3,5.1,2,9,3.5)

print(“Sum of all the numbers in tuple is:”, sorted(tuple1))

When we execute this code output is : Sum of all the numbers in the tuples is: 22.6

Built –in methods :

In python, tuples are immutable, its not allowed to change items of a tuple once it is assigned. There are only two tuple methods count() and index() that a tuple object can call.

count() : The count() method returns the number of times element occur in a tuple, means count method searches the given element in a tuple and returns how many times the element occurred in it.

The syntax of count() method is : tuple.count(element)

e.g. vowels = ('a', 'e', 'i', 'o', 'i', 'o', 'e', 'i', 'u')

count = vowels.count('i')

print('The count of i is:', count)

count = vowels.count('p')

print('The count of p is:', count)

When you run the above program, the output will be:

The count of i is : 3

The count of p is : 0

index() : The index() method searches an element in a tuple and returns its index. Index() method finds whether the given element present is tuple, if element is present it returns its

position and if not present it returns 0. However, if the same element is present more than once, the first/smallest position is returned.

The syntax of index() method for tuple is : index(element)

e.g. vowels = ('a', 'e', 'i', 'o', 'i', 'u')

index = vowels.index('e')

print('The index of e:', index)

index = vowels.index('i')

print('The index of i:', index)

index= vowels.index('p')

printf('The index of p:',index)

When you run the above program, the output will be:

The index of e is : 1

The index of i is : 2

ValueError: tuple.index(x): x not in tuple

Introduction to Dictionary

Dictionaries are a type of collection in Python. A dictionary is a collection which is unordered, changeable and indexed. Adictionary is a mapping between a set of indices (keys) and a set of values. It is an extremely useful data storage construct where each element is accessed by a unique key. In Python dictionaries are written with curly brackets, and they have keys and values. Keys are like addresses., but they don't have to be integers. They are usually characters. To create a dictionary, we use curly brackets. The keys are the first elements. They must be immutable and unique. Each key is followed by a value separated by a colon. The values can be immutable, mutable and duplicates. Each key and value pair is separated by a comma.

Properties of Dictionary Keys

There are two important points while using dictionary keys

- More than one entry per key is not allowed (no duplicate key is allowed)
- The values in the dictionary can be of any type while the keys must be immutable like numbers, tuples or strings.
- Dictionary keys are case sensitive- Same key name but with the different case are treated as different keys in Python dictionaries.

We can create a blank dictionary with no elements using following command.

```
blank_dict = {}
```

Consider the example of dictionary. The car features is the key and the value is its details.

```
dict1 = { "brand" : "Ford", "model" : "Mustang" , "year" : 1964}
print(dict1)
```

The output will be shown in next line

```
{'brand' : 'Ford', 'model' : 'Mustang', 'year' : 1964}
```

e.g. {'a' : 'apple', 'b': 'bill gates', 'c' : computer}

In the above example, we have created a list where each alphabet maps a English word i.e., keys are in characters (alphabets) and values are in strings.

I. CREATION OF DICTIONARY

We can create a dictionary using built-in function dict(), which creates a new dictionary with no items.

e.g.
```
A=dict()
print(A)
```
It prints {}

where {} represent empty string.

e.g.
```
new_dict = dict({1:'odd', 2:'even'})
```

It creates a dictionary with name new_dict.

Accessing values in dictionary

To access dictionary elements, square brackets along with the key is used to obtain its value.

e.g.
```
dict1 = {'Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science'}
print("dict['Name']: ", dict['Name'])
print("dict['Age']: ", dict['Age'])
```

When the above code is executed, it produces the following result –

```
dict['Name']: Sneha
dict['Age']: 23
```

e.g.
```
A=dict()
A['1']="one"
A['2']="two"
A['3']="three"
print A
```

When above code is executed , it produces the following result-

'1':'one','b':two','3':'three'

Traversing a dictionary

Traversing refers to visiting each element index at least one to access its value. This can be done using looping statement or by using for loop.

e.g. A={'Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science'}

```
for i in A:
   print i,":",A[i]
```

After running above code, the following result is displayed

Name : Sneha

Age : 21

Subject : Computer Science

1) *Update a Dictionary*

In Python dictionary objects are mutable means it is possible to add or update elements. You can push a new item or modify any existing with the help of the assignment operator. Whenever an element is added whose key already exists, it's value will get changed to the new value. On adding a fresh "key: value" pair, a new element will get added to the dictionary.

e.g. dict1 = {'Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science'}

Suppose the student change her subject from computer science to Mathematics

```
dict1['Subject'] = 'Mathematics'
print("dict1['Subject']: ", dict1['subject'])
```

The execution of the above code will produce the following output.

dict1['Subject']: 'Mathematics'

Suppose new element Class : TYBSc is added in dictionary as follows:

```
dict1['Class'] = 'TYBSc'
print(dict1)
```

The execution of above code will produce the following output.

{Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science', 'Class' : 'TYBSc'}

2) *Remove Elements from dictionary*

The most common method to remove element from dictionary is by using **"pop()"**method. It takes the key as the input and deletes the corresponding element from the Python dictionary. It returns the value associated with the input key. The other method "popitem()"removes and returns a random element (key, value) from the dictionary. All the elements from the dictionary can be remove using the **"clear"**method to flush everything. We can use "del" keyword to delete individual items and consequently the whole dictionary itself.

e.g. stud1 = {Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science', 'Class' : 'TYBSc'}

Suppose we have to delete a specific element

```
print(stud1.pop('Age'))
Output :  21
print(stud1)
```

It deletes key Age with its value:

```
{Name' : 'Sneha', 'Subject' : 'Computer Science', 'Class' : 'TYBSc'}
```

Suppose we have to delete a random element

```
print(stud1.popitem())
Output : ('Class','TYBSc')
print(stud1)
```

It deletes the arbitrary key Class with its value and print the result as follows :

```
{Name' : 'Sneha', 'Subject' : 'Computer Science'}
```

To delete all elements we can use clear method as follows :

```
dict1.clear()
print(stud1)
```

It gives the output as empty dictionary

```
{}
```

Now eliminate the dictionary object as follows :

```
del dict1
```

Now if we try to print the contents of dictionary dict1, it gives error

```
print(dict1)
```

Hence, the last print statement fails with the "NameError."

Built-in Dictionary functions:

len() : It returns the number of elements stored in the dictionary.

e.g. stud1 = {'Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science', 'Class' : 'TYBSc'}

print(len(stud1))

It returns the output value 4

all() : It returns true if all keys of dictionary are true (or if dictionary is empty).

e.g. squares : {0 : 0, 1 : 1, 3 : 9, 5 : 25, 7 : 49, 9 : 81}

print(all(squares))

It prints the output : False

any() : It returns true if any key of dictionary is true . If dictionary is empty , it returns false.

e.g. squares : {0 : 0, 1 : 1, 3 : 9, 5 : 25, 7 : 49, 9 : 81}

print(any(squares))

It prints the output : True

cmp() : Compares items of two dictionaries. It can return 3 possible values, i.e., 1, 0 or -1. If both dictionaries are equal, then it returns 0. If second dictionary have greater elements than first, then -1 and if first dictionary is greater than second then return 1. cmp() function was available in Python 2.x but is not available in Python 3.x .

e.g. dict1 = {'Name' : 'Roma', Age : 3}

dict2 = {'Name' : 'Suman', Age : 27}

dict3 = {'Name' : 'Mona', Age : 27}

dict4 = {'Name' : Roma', Age : 3}

print(cmp(dict1,dict2))

Output : -1

print(cmp(dict2,dict3))

Output : 1

print(cmp(dict1,dict4))

Output : 0

get() : returns the value of item with the specified key.

e.g. dict1 = {'Name' : 'Roma', 'Age' : 3}

x = dict1.get('Age')

print(x)

Output : 3

Python dictionary methods :

dict.clear() :This function deletes all elements of dictionary which makes dictionary empty.

stud1.clear()

print(stud1)

Output : {}

get() : This function returns a value for the given key. If key is not available then returns default value none.

dict.get(key, default=None) where key is to be searched in the dictionary.

e.g. dict = {'Name' : 'Mona', 'Age' : 23}

print(dict.get('Age'))

Output : 27

print(dict.get('Gender', "NA"))

Output : NA

dict.copy() : This method returns the copy of dictionary

dict1 = {'Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science', 'Class' : 'TYBSc'}

dict2 = dict1.copy()

print(dict2)

Output : {'Name' : 'Sneha', 'Age' : 21, 'Subject' : 'Computer Science', 'Class' : 'TYBSc'}

dict.values() : This function only display the values in dictionary not keys.

print(dict1.values())

Output : {"Sneha", 21, "Computer Science", "TYBSc"}

dict.keys() : This function is opposite to values(), it only display the keys in dictionary.

print(dict1.keys())

Output : {"Name", "Age", "Subject" , "Class" }

dict.items() : This function returns both keys and value with a representation, where both are well mapped.

e.g. dict1 = { 1:'one', 2:'two', 3:'three'}

print(dict1.items())

Output : { (1 : 'one'), (2 : 'two'), (3 : 'three')}

dict.fromkeys() : The method fromkeys() creates a new dictionary with keys from seq and values set to value.

dict.fromkeys(seq[, value])

Here seq is the list of values which would be used for dictionary keys preparation and value is optional, if provided then value would be set to this value.

e.g. seq = ('Name', Age', 'Gender')

dict = dict.fromkeys(seq)

print(str(dict))

Output : {'Name' : None, 'Age' : None, 'Gender' : None}

e.g. dict = dict.fromkeys(seq, 10)

print(str(dict))

Output : {'Name' : 10, 'Age' : 10, 'Gender' : 10}

3) *sorted() : Sorted function allows to sort data either by the key or the value .*

e.g. dict1 = { 1: 'one' , 3 : 'three', 2 : 'two', 4 : 'four'}

print(sorted(dict1.keys()))

Output : [1,2,3,4]

print(sorted(dict1.items()))

Output : [(1: 'one', 2:'two', 3:'three', 4:'four')]

print(sorted(dict1.values()))

Output:['four', 'one' , 'three' , 'two']

Objective type questions for 1 mark

1. Which of the following creates a tuple?

A. tuple1=("a","b")

B. tuple1[2]=("a","b")

C. tuple1=(5)*2

D. None of the above

2. Choose the correct option with respect to Python.

A. Both tuples and lists are immutable.

B. Tuples are immutable while lists are mutable.

C. Both tuples and lists are mutable.

D. Tuples are mutable while lists are immutable.

3. Choose the correct option.

A. In Python, a tuple can contain only integers as its elements.

B. In Python, a tuple can contain only strings as its elements.

C. In Python, a tuple can contain both integers and strings as its elements.

D. In Python, a tuple can contain either string or integer but not both at a time.

4. Which of the following is correct with respect to d={"a":3,"b":7}

A. dictionary d is created.

B. a and b are the keys of dictionary d.

C. 3 and 7 are the values of dictionary d

D. All of the above.

5. Which one of the following is correct?

A. In python, a dictionary can have two same keys with different values.

B. In python, a dictionary can have two same values with different keys

C. In python, a dictionary can have two same keys or same values but cannot have two same key-value pair

D. In python, a dictionary can neither have two same keys nor two same values.

6. What will be the output of above Python code

```
d1={"abc" : 5,"def" : 6,"ghi" : 7}
print(d1[0])
```

A. abc

B. 5

C. {"abc":5}

D. Error

7. What is output of given code

```
dict={"Joey":1,"Rachel":2}
dict.update({"Phoebe":2})
        print(dict)
```

A. {"Joey":1,"Rachel":2,"Phoebe":2}

B. {"Joey":1,"Rachel":2}

C. {"Joey":1,"Phoebe":2}

D. Error

8. Which of the following is False regarding dictionary in Python?

A. Values of a dictionary can be string,integers or combination of both.

B. Keys of a dictionary can be string,integers or combination of both

C. The value of a dictionary can be accessed with the help of indices.

D. None of the above

9. Which of the following will delete key_value pair for key="tiger" in dictionary?

dic={"lion":"wild","tiger":"wild","cat":"domestic","dog":"domestic"}

A. **del dic["tiger"]**

B. dic["tiger"].delete()

C. delete(dic.["tiger"])

D. del(dic.["tiger"])

10.
```
t1 = (1,2,3)
t2 = (4,5,6)
t3 = t1 + t2
print(t3)
```

A. (1,2,3,4,5,6)

B. (1,2,3,1,2,3)

C. (4,5,6,4,5,6)

D. None of these

11. tuple1 = (11,22,33,44,55,66)

print(len(tuple1)

A. 7

B. 6

C. 5

D. None of these

12. vowels = ('a', 'e', 'i', 'o', 'i', 'o', 'e', 'i', 'u')

count = vowels.count('o')

print(count)

A. 2

B. 3

C. 0

D. 1

13. Is the following Python code valid?

a,b=1,2,3

A. Yes, this is an example of tuple unpacking. a=1 and b=2

B. Yes, this is an example of tuple unpacking. a=(1,2) and b=3

C. No, too many values to unpack

D. Yes, this is an example of tuple unpacking. a=1 and b=(2,3)

14. What will be the output of the following Python code?

>>> a=(1,2)>>> b=(3,4)>>> c=a+b>>> c

A. (4,6)

B. (1,2,3,4)

C. Error as tuples are immutable

D. None

15. Tuples can't be made keys of a dictionary.

A. True

B. False

C.Somewhat true

D.None of these

16. What will be the output of the following Python code?

>>> a=(2,3,1,5)>>> print(sorted(a))

A. **[1,2,3,5]**

B. (2,3,1,5)

C. (1,2,3,5)

B. None of these

17. Is the following Python code valid?

>>> a=(1,2,3)>>> b=a.update(4,)

A. Yes, a=(1,2,3,4) and b=(1,2,3,4)

B. Yes, a=(1,2,3) and b=(1,2,3,4)

C. No because tuples are immutable

D. No because wrong syntax for update() method

18.vowels = ('a', 'e', 'i', 'o', 'i', 'u')

index = vowels.index('e')

A.0

B.1

C.2

D.None of these

19.dict1 = { 11: 'Eleven' , 13 : 'Thirteen', 12 : 'Twelve', 14 : 'Fourteen'}

print(sorted(dict1.keys()))

A. [11,12,13,14]

B. ['Eleven','Twelve','Thirteen','Fourteen']

C. [14,13,12,11]

D. None of these

20. print(sorted(dict1.values()))

A.[11,12,13,14]

B. ['Eleven', 'Fourteen','Thirteen','Twelve']

C. [14,13,12,11]

D. None of these

Questions of 1 mark

1.What is output of t = (1,2,3,4)

t[1:3]

2.What is output of

D = {"Rohan":40, "Sohan":45}

D["Rohan"]

3.What is output of

t = (1, 2)

2* t

4.What is output of

My_tup = (1, 2, 3, 4)

My_tup.append(5, 6, 7)

print len(My_tup)

5. student = {1: {'name' : 'Seema', 'age' : '27'}, 2: {'name': 'Reena', 'age' : '30'}}

What is correct command to print Reena's age?

6. Select the correct way to **remove** the key '**marks**' from a dictionary.

Student = {'name' : 'Sohan', 'class' : 'FYBSc', "marks" : 75}

Questions of 2 marks

1. The dictionary d ={0:10,1:20}. Write correct code to add key- value pair 2:30 in dictionary d.

2. Write a Python script to concatenate following dictionaries to create a new one

dic1={1:10, 2:20}

dic2={3:30, 4:40}

dic3={5:50,6:60}

3. How to delete tuples?

4. How to update tuples?

5. Explain cmp(tuple1,tuple2) with example.

6. Explain any two built-in functions with example.

7. Explain any two built-in methods with example.

Questions of 4 marks

1.Write any 4 built-in functions with example.

2.Explain any 4 built-in methods with example.

3. Write a Python program to sum all the items in a dictionary.

4. Write a Python script to merge two Python dictionaries.

5. Write a Python program to remove a key from a dictionary.

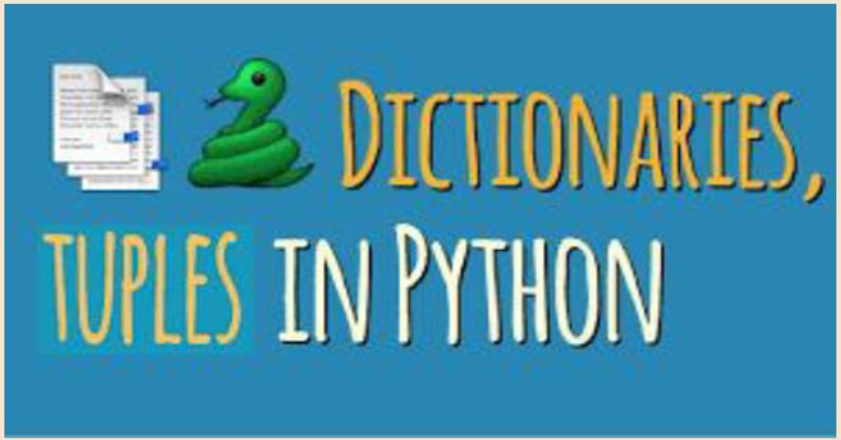
6. Write a Python program to get the maximum and minimum value in a dictionary.

Q.1	What is output of t = (1,2,3,4) t[1:3]
Q.1	What is output of D = {"Rohan":40, "Sohan":45} D["Rohan"]
Q.1	What is output of t = (1, 2) 2* t
Q.1	What is output of My_tup = (1, 2, 3, 4) My_tup.append(5, 6, 7) print len(My_tup)
Q.1	student = {1: {'name' : 'Seema', 'age' : '27'}, 2: {'name': 'Reena', 'age' : '30'}} What is correct command to print Reena's age?

Q.1	Select the correct way to **remove** the key '**marks**' from a dictionary. Student = {'name' : 'Sohan', 'class' : 'FYBSc', "marks" : 75}
Q.1	How empty tuple is represented?
Q.1	What is output of following code? ttuple = ("Sayara", 156, 57.8) print(ttuple[-1])
Q.1	What is the use of all()?
Q.1	Which commands are used to eliminate dictionary object ?
Q.1	How to sort the dictionary data either by keys or by values?
Q.2	The dictionary d ={0:10,1:20}. Write correct code to add key- value pair 2:30 in dictionary d..
Q.2	Write a Python script to concatenate following dictionaries to create a new one dic1={1:10,2:20} dic2={3:30,4:40} dic3={5:50,6:60}
Q.2	How to delete tuples?
Q.2	How to update tuples?
Q.2	Explain cmp(tuple1,tuple2) with example.
Q.2	Explain any two built-in tuple functions with example.
Q.2	Explain any two built-in tuple methods with example.
Q.2	Explain any two built-in dictionary functions with example.
Q.2	Explain any two built-in dictionary methods with example.
Q.2	Explain get() in dictionary with example.
Q.3	How to create a tuple and print its contents, explain with example.
Q.3	How to access elements of tuple using negative indexing? Explain with example.
Q.3	Create tuple1 with contents 1,2,3 and tuple2 with contents 'x', 'y', 'z' . Now concatenate these two tuples into tuple3. Write python script.
Q.3	Suppose tuple1=(2,4,6) . Replicate the data in the tuple with 3 times, using python commands.
Q.3	Suppose tuple countries = ("USA", "India", "London", "Singapore"). Check whether country "India" is member of tuple countries using python coding.
Q.3	What is dictionary? How to create and print dictionary content? Explain with example.
Q.3	What is mean by key and value pair in dictionary? Explain with example.
Q.3	What is the purpose of count built-in tuple method?
Q.3	How to update a dictionary? Explain with example.

Q.3	What is purpose of cmp() built-in function in dictionary ?
Q.4	How to copy one dictionary to another? Explain with example.
Q.4	What are the major advantages of tuple?
Q.4	What is the difference between list and tuple?
Q.4	What is the difference between list and dictionary?
Q.4	Write a code for creating a tuple with different data types and accessing elements of same tuple with suitable example.
Q.4	Write a Python program to add an item in a tuple
Q.4	Write a Python program to get the 4th element and 4th element from last of a tuple with suitable example.
Q.4	Write a Python program to sum all the items in a dictionary.
Q.4	Write a Python script to merge two Python dictionaries.
Q.4	Write a Python program to remove a key from a dictionary.
Q.5	Write a Python program to reverse contents of a tuple.
Q.5	Write a Python program to get the maximum and minimum value in a dictionary
Q.5	Write a program to create two tuples with names tup1 and tup2 and then perform concatenation and repetition operation on tuples and print the new tuples.
Q.5	Write a program to create a tuple and then perform membership operation on tuple.
Q.5	Explain any three built-in tuple functions with example.
Q.5	Explain any three built-in tuple methods with example.
Q.5	Explain any three built-in dictionary functions with example.
Q.5	Explain any three built-in dictionary methods with example.
Q.5	Write a program to create a dictionary which contain key value pair as name of student and his/her age. Sort the contents of dictionary with 'key' and with 'value'
Q.5	Write a program to create a dictionary of employee information and then traverse the contents of dictionary with key and with value.

Unit-4 Python Tuples and Dictionary



Prof. Dr. B. H.Barhate

Python Tuples

- A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]), and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists.

Creating Tuples

To create a tuple in Python, place all the elements in a () parenthesis, separated by commas. A tuple can have heterogeneous data items, a tuple can have string and list as data items as well.

e.g. tuple1 = (“Sameer”, 156, 57.8)

tuple2 = (1,2,3,4,5)

tuple3 = (“apple”, “watermelon”, “banana”)

If we want to display contents of tuple3, we give the command

print(tuple3)

The output is : (‘apple’,’watermelon’,’banana’)

We can also create a tuple from unpacked elements i.e without any braces just elements separated by comma i.e.

e.g. tup1 = “Seema”, 23 , “Dhule”, 49.33

print(tup1)

Output : (‘Seema’,23,’Dhule’,49.33)

Deleting Tuples

Tuples are unchangeable , so you cannot remove items from tuple completely. It is not possible to remove individual tuple elements but you can delete the tuple completely by using del command:

e.g. tuple1 contain name of person with its class and marks of 3 subjects

```
tuple1 = (“Sameer”,“TYBSc”,76,78,85)
print(tuple1)
(‘Sameer’, ‘TYBSC’, 76,78,85)
del tuple1
print(tuple1)
```

If we delete a tuple and then print the tuple , it gives the error message as

NameError: name 'tuple1' is not defined

Accessing Tuples

Elements of tuple can be access by referring to the index number, inside square brackets.

e.g. thistuple = (“Sameer”, 156, 57.8)

print(thistuple[0])

When above code is executed it produces the following result

Sameer

If we want to print the contents of tuple at index 1, it should be as follows

print(thistuple[1])

156

We can also access the elements using negative indexing. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

e.g. thistuple = (“Sameer”, 156, 57.8)

print(thistuple[-1])

When we execute the above code, it displays the last element of the tuple, which is at -1 index.

57.8

print(thistuple[-2])

Cont.

Elements of tuple can be access by referring to the index number, inside square brackets.

e.g. thistuple = (“Sameer”, 156, 57.8)

print(thistuple[0])

When above code is executed it produces the following result

Sameer

If we want to print the contents of tuple at index 1, it should be as follows

print(thistuple[1])

156

We can also access the elements using negative indexing. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

e.g. thistuple = (“Sameer”, 156, 57.8)

print(thistuple[-1])

When we execute the above code, it displays the last element of the tuple, which is at -1 index.

57.8

print(thistuple[-2]) Result is 156

Tuple Operations

Tuples Operations : There are some operations perform on tuple as follows :

Concatenation: Tuples can be concatenated by using '+' operator, the result is new tuple and not a string.

e.g. tuple1 = (1,2,3)

tuple2 = ("x","y","z")

tuple3 = tuple1 + tuple2

print(tuple3)

It concatenates the tuple1 and tuple2 and produces the result as new tuple

(1,2,3,,'x','y','z')

Repetition : Sometime we need to replicate the data in a tuple. The multiplication operator is used for repetition to make multiple copies of a tuple and joins them all together.

```
tup1 =(1,2,3)

re_tupl = (tup1,) * 2

print(re_tup1)
```

It replicates the contents of tuple tupl1 and prints the result as a new tuple

(1,2,3,1,2,3)

Membership : It is used to determine if a specified item is present in a tuple. Membership in a tuple is checked using the 'in' keyword:

```
Membership test in tuple

my_tuple = ('a', 'p', 'p', 'l', 'e',)

# In operation

print('a' in my_tuple)

print('b' in my_tuple)

# Not in operation

print('g' not in my_tuple)
```

Iteration : We can iterate over tuples using a simple for loop. Tuples are **sequence data type** and we can iterate over sequences.

```
# Using a for loop to iterate through a tuple

for name in ('Ranjan', 'Nita'):

    print("Hello", name)
```

Output

```
Hello Ranjan

Hello Nita
```

Built-in Tuples functions

Following are built in tuple functions :

len( ) : len( ) is a buit-in tuple function to compute number of items present in given.

e.g. fruit_names = ("apple", “watermelon”, "banana")

print(len(fruit_names))

The result is : 3

cmp(tuple1,tuple2) : cmp( ) is a built-in tuple function in which elements of tuple1 are compared with elements of tuple2 , if both lists are same it returns 0 otherwise it returns 1.

e.g. tuple1 = (‘x’,y’,’z’)

tuple2 = (1,2,3)

tuple3 = (‘x’,y’,’z’)

cmp(tuple1,tuple2)

output : 1

cmp(tuple1,tuple3)

output : 0

any() : This built-in tuple function returns true if any element present in a tuple and returns false if tuple is empty.

e.g. tuple1 = (‘x’,y’,’z’)

print(“Is there any element is present in tuple :”, any(tuple1))

Output : Is there any element is present in tuple : True

e.g. tuple2 = ( )

print(“Is there any element is present in tuple :”, any(tuple2))

Output : Is there any element is present in tuple : False

min( ) : This tuple function returns smallest element(int) of the tuple

e.g. tuple1=(4,5.7,2.3,5)

print("Smallest element in tuple is :", min(tuple1))

It prints the result as : Smallest element in tuple is : 2.3

max( ) : This tuple function returns largest element(int) of the tuple

e.g. tuple1=(4,5.7,2.3,5)

print("Largest element in tuple is :", max(tuple1))

It prints the result as : Largest element in tuple is : 5.7

sorted( ) : It is used to sort all elements of the tuple.

e.g. tuple1=(4,5.7,2.3,5)

print("Sorted elements in tuple are :", sorted(tuple1))

It prints the result as : Sorted element in the tuple is : (2.3,4,5,5.7)

e.g. tuple1 = ('e', 'a', 'u', 'o', 'i')

print("Sorted characters are:", sorted(tuple1))