JAYAM ARTS AND SCIENCE COLLEGE

DEPARTMENT OF COMPUTERSCIENCE

COURSE	:	III B.SC -CS
PAPERCODE	:	21UCS10
PAPER NAME	:	PROGRAMMING IN PYTHON
SEMESTER	:	VI

Subject Title	**PROGRAMMINGIN PYTHON**	**Semester**	**VI**
Subject Code	**21UCS10**	**Specialization**	**NA**
Type	**Core:Theory**	**L:T:P:C**	**86:6:0:5**
Unit	**Contents**	**Levels**	**Session s**
I	Python – origins – features – variable and assignment - Python basics – statement and syntax – Identifiers – Basic style guidelines–Pythonobjects–Standardtypesandotherbuilt-in types – Internaltypes – Standard type operators –Standard type built-in functions.	**K1**	**13**
II	Numbers – Introduction to Numbers – Integers – Double precision floating point numbers – Complex numbers – Operators–Numerictypefunctions–Sequences:Strings,Lists and Tuples – Sequences – Strings and strings operators – String built-in methods – Lists – List type Built in Methods – Tuples.	**K2**	**13**
III	Mapping type: Dictionaries – Mapping type operators –Mapping type Built-in and Factory Functions - Mapping type built in methods – Conditionals and loops – if statement – else Statement – elif statement – conditional expression – while statement – forstatement – break statement –continue statement – pass statement – Iterators and the iter( ) function - Files and Input/Output–Fileobjects–Filebuilt-infunctions–Filebuilt-in methods – File built-in attributes – Standard files – command line arguments.	**K3**	**20**
IV	Functionsand FunctionalProgramming – Functions–calling functions – creating functions – passing functions – Built-in Functions:apply(),filter(),map()andreduce()-Modules–Modules and Files – Modules built-in functions - classes – class attributes – Instances.	**K4**	**20**
V	Database Programming – Introduction - Basic Database OperationsandSQL-ExampleofusingDatabaseAdapters, Mysql- Regular Expression –SpecialSymbolsandCharacters – REs and Python.	**K5**	**20**
	LearningResources		
Text Books	TitleofBookPublisherYearofPublication1WesleyJ.ChunCore Python Programming Pearson Education Publication 2012		
Reference Books	1. WesleyJ.ChunCorePythonApplicationProgrammingPearsonEducation Publication 2015 2. EricMatthesPythoncrashcourseWilliampollock 2016 3. ZedShawLearnPythonthehardwayAdditionWesley2017 4. MarkLutzPythonpocketreferenceO’ReillyMedia2014Pedagogy		
Website/ Link	1. https://www.tutorialspoint.com/python/ 2. www.spoken-tutorial.org		

UNIT-I

Python

Today,Pythonisoneofthemostpopularprogramminglanguages.Althoughitisa general-purpose language, it is used in various areas of applications such as Machine Learning, Artificial Intelligence, web development, IoT, and more.

Whatis Python?

Pythonisaverypopulargeneral-purposeinterpreted,interactive,object-oriented, andhigh-levelprogramminglanguage.Pythonisdynamically-typedandgarbage-collected programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Pythonsupportsmultipleprogrammingparadigms,includingProcedural,ObjectOriented and Functional programming language. Python design philosophyemphasizes code readability with the use of significant indentation.

someofthe**keyadvantages**oflearning Python:

- **PythonisInterpreted**−Pythonisprocessedatruntimebytheinterpreter.You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **PythonisInteractive**−YoucanactuallysitataPythonpromptandinteract with the interpreter directly to write your programs.
- **PythonisObject-Oriented**−PythonsupportsObject-Orientedstyleor technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-levelprogrammersandsupportsthedevelopmentofawiderangeof applications from simple text processing to WWW browsers to games.

CharacteristicsofPython:-

Followingareimportantcharacteristicsof**PythonProgramming**−

- ItsupportsfunctionalandstructuredprogrammingmethodsaswellasOOP.
- Itcanbeusedasascriptinglanguageorcanbecompiledtobyte-codefor building large applications.

- Itprovidesveryhigh-leveldynamicdatatypesandsupportsdynamictype checking.
- Itsupportsautomaticgarbagecollection.
- ItcanbeeasilyintegratedwithC,C++,COM,ActiveX,CORBA,andJava.

ApplicationsofPython:-

ThelatestreleaseofPythonis3.x.Asmentionedbefore,Pythonisoneofthemost widely used language over the web. I'm going to list few of them here:

- **Easy-to-learn**−Pythonhasfewkeywords,simplestructure,andaclearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read**−Pythoncodeismoreclearlydefinedandvisibletotheeyes.
- **Easy-to-maintain**−Python'ssourcecodeisfairlyeasy-to-maintain.
- **Abroadstandardlibrary**−Python'sbulkofthelibraryisveryportableand cross-platform compatible on UNIX, Windows, and Macintosh.
- **InteractiveMode**−Pythonhassupportforaninteractivemodewhichallows interactive testing and debugging of snippets of code.
- **Portable**−Pythoncanrunonawidevarietyofhardwareplatformsandhasthe same interface on all platforms.
- **Extendable**−Youcanaddlow-levelmodulestothePythoninterpreter.These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases**−Pythonprovidesinterfacestoallmajorcommercialdatabases.
- **GUIProgramming**−PythonsupportsGUIapplicationsthatcanbecreated and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable**−Pythonprovidesabetterstructureandsupportforlargeprograms than shell scripting.

ORIGINS

GuidoVanRossum,aDutchprogrammer,createdPythonprogramminglanguage.In the late 80's, he had been working on the development of ABC language in a computer science research institute named **Centrum Wiskunde & Informatica**(CWI)intheNetherlands.In1991,VanRossumconceivedand published Python as a successor of ABC language.

Formanyuninitiatedpeople,thewordPythonisrelatedtoaspeciesofsnake.Rossum though attributes the choice of the name Python to a popular comedy series **Monty Python's Flying Circus** on BBC.

Being the principal architect of Python, the developer community conferred upon himthe title of **Benevolent Dictator for Life** (BDFL). However, in 2018, Rossum relinquished the title. Thereafter, the development and distribution of the reference implementationofPythonishandledbyanonprofitorganization**PythonSoftware Foundation**.

MajorPythonReleases

Followingaretheimportantstagesinthehistoryof Python

Python0.9.0

Python'sfirstpublishedversionis0.9.ItwasreleasedinFebruary1991.It consisted of support for core object-oriented programming principles.

Python 1.0

InJanuary1994,version1.0wasreleased,armedwithfunctionalprogramming tools, features like support for complex numbers etc.

Python 2.0

Next major version − Python 2.0 was launched in October 2000. Many new featuressuchaslistcomprehension,garbagecollectionandUnicodesupportwere included with it.

Python 3.0

Python3.0,acompletelyrevampedversionofPythonwasreleasedinDecember 2008.Theprimaryobjectiveofthisrevampwastoremovealotofdiscrepancies that had crept in Python 2.x versions. Python 3 was backported to Python 2.6. It alsoincludedautilitynamedas **python2to3**tofacilitateautomatictranslationof Python 2 code to Python 3.

EOLforPython 2.x

Even after the release of Python 3, Python Software Foundation continued to supportthePython2branchwithincrementalmicroversionstill2019.However,it decided to discontinue the support by the end of year 2020, at which time Python 2.7.17wasthelastversioninthebranch.

CurrentVersion

More and more features have been incorporated into Python's 3.x branch. As ofdate,Python3.11.2isthecurrentstableversion,releasedinFebruary2023.

What'sNewinPython 3.11?

One of the most important features of Python's version 3.11 is the significant improvementinspeed.AccordingtoPython'sofficialdocumentation,thisversion is faster than the previous version (3.10) by up to 60%. It also states that the standard benchmark suite shows a 25% faster execution rate.

- Python3.11hasabetterexceptionmessaging.Insteadofgeneratingalong traceback on the occurrence of an exception, we now get the exact expression causing the error.
- AsperthereccommendationsofPEP678,the **add_note()**methodisaddedto the BaseException class. You can call this method inside the except clause and pass a custom error message.
- Italsoaddsthe**cbroot()**functioninthe**maths**module.Itreturnsthecube root of a given number.
- A new module **tomllib** is added in the standard library. TOML (Tom's ObviousMinimalLanguage)canbeparsedwithtomlibmodulefunction.

FEATURES

Pythonisafeaturerichhigh-level,interpreted,interactiveandobject-oriented scripting language.

Some of the important features of Python that make it widely popular. Apart from these10featureswherearenumberofotherinterestingfeatureswhichmakePython most of the developer's first choice.

1.Easyto learn

2. Interpreterbased

3.Interactive

4.Multi-paradigm

1. Largestandardlibrary

2. Opensource&crossplatform

3. GUIdevelopment

4. Database connectivity

9.Extensible10.Developer

community

PythonisEasyto Learn

OneofthemostimportantreasonsforthepopularityofPython.Pythonhasalimited set of keywords. Its features such as simple syntax, usage of indentation to avoid clutterofcurlybracketsanddynamictypingthatdoesn'tnecessitatepriordeclaration of variable help a beginner to learn Python quickly and easily.

PythonisInterpreterBased

Instructionsinanyprogramminglanguagesmustbetranslatedintomachinecodefor the processor to execute them. Programming languages are either compiler based or interpreter based.

In case of a compiler, a machine language version of the entire source program is generated.Theconversionfailsevenifthereisasingleerroneousstatement. Hence, the development process is tedious for the beginners. The C family languages (including C, C++, Java, C Sharp etc) are compiler based.

Python is an interpreter based language.Theinterpreter takesoneinstruction fromthe source code at a time, translates it into machine code and executes it. Instructions beforethefirstoccurrenceoferrorareexecuted.Withthisfeature,itiseasiertodebug the program and thus proves useful for the beginner level programmer to gain confidence gradually. Python therefore is a beginner-friendly language.

PythonisInteractive

Standard Python distribution comes with an interactive shell that works on the principle of REPL (Read – Evaluate – Print – Loop). The shell presents a Python prompt >>>. You can type any valid Python expression and press Enter. Python interpreterimmediatelyreturnstheresponseandthepromptcomesbacktoreadthe next expression.

```
>>>2*3+1
7
>>>print("HelloWorld")
HelloWorld
```

The interactive mode is especially useful to get familiar with a library and test out its functionality.Youcantryoutsmallcodesnippetsininteractivemodebeforewritinga program.

PythonisMultiParadigm

Python is a completely object-oriented language. Everything in a Python program is anobject.However,Pythonconvenientlyencapsulatesitsobjectorientationtobeused as an imperative or procedural language – such as C. Python also provides certain functionality that resembles functional programming. Moreover, certain third-party tools have been developed to support other programming paradigms such as aspect-oriented and logic programming.

Python'sStandardLibrary

It has a very few keywords (only Thirty Five), Python software is distributed with a standardlibrarymadeoflargenumberofmodulesandpackages.ThusPythonhasout of box support for programming needs such as serialization, data compression, internet data handling, and many more. Python is known for its batteries included approach.

PythonisOpenSourceandCrossPlatform

Python'sstandarddistributioncanbedownloaded fromhttps://www.python.org/downloads/withoutanyrestrictions.Youcandownload pre-compiled binaries for various operating system platforms. In addition, the source code is also freely available, which is why it comes under open source category.

Pythonsoftware(alongwiththedocumentation)isdistributedunderPythonSoftware Foundation License. It is a BSD style permissive software license and compatible to GNU GPL (General Public License).

Python is a cross-platform language. Pre-compiled binaries are available for use on variousoperatingsystemplatformssuchasWindows,Linux,MacOS,AndroidOS. The reference implementation of Python is called CPython and is written in C. You can download the source code and compile it for your OS platform.

A Python program is first compiled to an intermediate platform independent byte code. The virtual machine inside the interpreter then executes the byte code. This behaviourmakesPythonacross-platformlanguage,andthusaPythonprogramcanbe easily ported from one OS platform to other.

PythonforGUI Applications

Python's standard distribution has an excellent graphics library called TKinter. It is a Python port for the vastly popular GUI toolkit called TCL/Tk. You can build attractiveuser-friendlyGUIapplicationsinPython.GUItoolkitsaregenerallywritten in C/C++. Many of them have been ported to Python. Examples are PyQt, WxWidgets, PySimpleGUI etc.

Python'sDatabaseConnectivity

Almost any type of database can be used as a backend with the Python application. DB-API is a set of specifications for database driver software to let Python communicatewitharelationaldatabase.Withmanythirdpartylibraries,Python can also work with NoSQL databases such as MongoDB.

PythonisExtensible

The term extensibility implies the ability to add new features or modify existing features. As stated earlier, CPython (which is Python's reference implementation) is written in C. Henceone can easily write modules/libraries in C and incorporate them in the standard library. There are other implementations of Python such as Jython (writteninJava)andIPython(writteninC#).Hence,itispossibletowriteandmerge new functionality in these implementations with Java and C# respectively.

Python'sActiveDeveloperCommunity

AsaresultofPython'spopularityandopen-sourcenature,alargenumberofPython developers often interact with online forums and conferences. Python Software

Foundationalsohasasignificantmemberbase,involvedintheorganization'smission to "**Promote, Protect, and Advance the Python Programming Language**"

Pythonalsoenjoysasignificantinstitutionalsupport.MajorITcompaniesGoogle, Microsoft, and Meta contribute immensely by preparing documentation and other resources.

VARIABLEANDASSIGNMENT

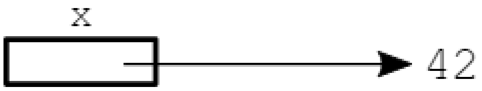
PythonVariables

APythonvariableisanamedbitofcomputermemory,keepingtrackofavalueasthe code runs.

Avariableiscreatedwithan"assignment"equalsign=,withthevariable'snameon the left and the value it should store on the right:

```
   x=42
```

Inthecomputer'smemory,eachvariableislikeabox,identifiedbythenameofthe variable. In the box is a pointer to the current value for that variable.



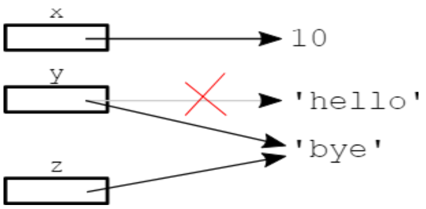
Laterinthecode,appearancesofthatvariablename,e.g. x,retrieveitscurrentvalue, in thiscase42.Theuseofthevariablenamein thecodedoesnothavequotesaround it or anything. The variable name x is just a bare word in the code.

Tryingtoretrievethevalueofavariablethatdoesnotexistfailswithanerror(i.e. no = ever assigned that variable name).

VariableAssignmentRules

Hereisamorecomplicatedcodeexampleandapictureofmemoryafterthiscode runs.

```
   x=10
   y='hello'
   y='bye'
   z= y
```



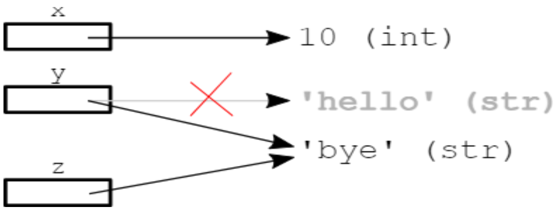
1. Theassignmentx=10simplysetsxtopointto 10.

2. Theassignmenty='hello' setsytopointto'hello'.Thentheliney= 'bye' changes y to point to 'bye', overwriting the first pointer. Assigning a variable overwrites any existing pointer that variable had. Each assignment is like the phrase "nowpointto"—thevariablenowpointstothenewthing,andanyprevioussettingis forgotten.

3. Assignmentbetween two variables like z= y, sets z to point to the same thing as y. Now they both point to the same value. It does not set one variable to point to the other variable, although the code does kind of look like that. It also does not set up a permanentrelationshipbetweenthevariables,liketheymustalwaysbethesamenow. Confusingly, in mathematics writing the symbol = does set up a permanent relationship. In code, z = y has avery limited meaning: set z to point to what y points to at this moment.

EveryValuehasaType

Hereisthesamepictureasabove,butwithmoredetailadded.



In Python, every value in memory is tagged with its "type" - so we see the integer 10hasalittle (int) offto its side —intis thenameoftheinteger typein Python.Thestring 'hello' istaggedwith strwhichisthenameofthestringtype.

AsPythonruns,manyoperationsdependonthisfeature,treatingavalueappropriately depending on its type. See here how the + operator behaves differently if it is givenintvs.strvalues:

```
>>>1 + 2        #intvalues
3
>>>'a'+'b'      # strvalues
```

```
'ab'
>>>'3' + '4'      #strthatlooklikeint
'34'
```

MemoryandtheGarbageCollector

Thestring'hello'intheexampleaboveisshowningray.Itisnotneededbythecode after the third line runs — no variable points to it any longer, so it cannot be used. Memorylikethis,whichisnolongeraccessible,iscalled"garbage"incomputercode. A "garbage collector" is a system that reclaims garbage memory, such as 'hello' here, so its memory can be re-used to hold a new value. This is something Python does automatically behind the scenes. The garbage collector slows the running of the code down a little.

Many modern languages have a garbage collector to reclaim garbage memory automatically. A few languages instead make the programmer identify garbage memory on their own - this has the potential to run fast, but it is a chore for the programmerandabigsourceofbugswhentheprogrammermis-identifiesgarbage memory.ThedesignofPythonprioritizesprogrammerproductivity,soitisnatural that Python includes a garbage collector.

VariableSwap

Wehavetwovariablesandwewantto"swap"theirvalues,soeachtakesonthevalue of the other.

Thisisalittlecodingmovethatallprogrammersshouldknow.

```
a= 42
b=13
```

Itmightseemthatonecanbegin witha=b,butthisdoesnotwork,sinceitoverwrites and thus loses the original value of a. The classic 3-line solution uses a temporary variable named "temp" to hold this value during the swap, like this:

```
temp=a
a= b
b= temp
```

Startingwiththeabovediagram,youcantracethroughthethreeassignments,leading to this memory structure

VariableNamesareSuperficialLabels

Normallyvariablenamesarechosentoreflectwhatdatatheycontain.Thatsaid,there is one funny feature of variable names in code.

Considerthefollowingcomputation

```
>>>x=6
>>>y=x+x
>>>y
12
```

Usingacouplevariables,itcomputesthatdoubling6makes12.Supposeinsteadit was written this way:

```
>>>alice=6
>>>bob=alice+alice
>>>bob
12
```

PYTHON

BASICSSTATEMENTANDS

YNTAXSTATEMENT

Anyinstructionwritteninthesourcecodeandexecutedbythe Pythoninterpreter is calledastatement.ThePythonlanguagehasmanydifferenttypesofstatementslike assignment statements, conditional statements, looping statements, etc., that help a programmer get the desired output.

Forexample,p=9;isanassignmentstatement.

Python Statement

Aconditionalstatementisalogicalexpressionwhereoperatorscompare,evaluate,or checkiftheinputmeetstheconditionsandreturns‘True’.Ifyes,theinterpreterexecutes aspecificsetofinstructions.Ontheotherhand,loopingstatementsrepeatedlyexecutea setofinstructionsaslongasthedefinedconditionsaremetorsatisfied.

MultilineStatements

Usually,weusethenewlinecharacter(/n)toendaPythonstatement.However,ifyou wanttoexpandyourcodeovermultiplelines,likewhenyouwanttodolong calculationsandcan’tfityourstatementsintooneline,youcanusethecontinuation character (/).

```
COPYCODE


  s=10+15+30+ \
    49+5 +57 + \
    3- 54-2
```

Anotherwaytocreatemultilinestatementsistouseparentheses(),braces{},square brackets[],orevenasemi-colon(;).Whilethecontinuationcharactermarksanobvious linecontinuation,theothermultilinemethodsimplycontinuationindirectly.

```
COPYCODE


s=(10+15+30+
   49+5 +57 +
   3- 54-2)

p=['Hello',
'welcome',
'to Python']

x =5;p= 83;o= 7
```

Weusesemicolonstoplacemultiplestatementsinoneline.

SYNTAX

syntax defines a set of rules that are used to create a Python Program. The Python Programming Language Syntax has many similarities to Perl, C, and Java ProgrammingLanguages.However,therearesomedefinitedifferencesbetweenthe languages.

Itsupportsmultipleprogrammingparadigms,includingstructured,object-orientedprogramming,and functionalprogramming,and boasts adynamictypesystemand automatic memory management.

Python's syntaxis simple and consistent, adhering to the principle that "There should beone—andpreferablyonlyone —obviouswaytodoit."Thelanguageincorporates built-in data types and structures, control flow mechanisms, first-class functions, and modules for better codereusability and organization. Pythonalso usesEnglishkeywordswhereotherlanguagesusepunctuation,contributingtoits uncluttered visual layout.

Thelanguageprovidesrobusterrorhandlingthroughexceptions,andincludes a debuggerin the standard library for efficient problem-solving. Python's syntax, designedforreadabilityandeaseofuse,makesitapopularchoiceamongbeginners and professionals alike.

ExecutePythonSyntax

Aswelearnedinthepreviouspage,Pythonsyntaxcanbeexecutedbywritingdirectly in the Command Line:

```
>>>print("Hello,World!")
Hello, World!
```

Orbycreatingapythonfileontheserver,usingthe.pyfileextension,andrunningit in the Command Line:

```
C:\Users\YourName>pythonmyfile.py
```

PythonIndentation

Indentationreferstothespacesatthebeginningofacodeline.

Whereinotherprogramminglanguagestheindentationincodeisforreadabilityonly, the indentation in Python is very important.

Pythonusesindentationtoindicateablockofcode.

Example

```
if5 >2:
  print("Fiveisgreaterthantwo!")
```

giveyouanerrorifyouskiptheindentation:

Example

SyntaxError:

```
if5 >2:
print("Fiveisgreaterthantwo!")
```

Thenumberofspacesisuptoyouasaprogrammer,themostcommonuseisfour,but it has to be at least one.

Example

```
if5 >2:
 print("Fiveisgreaterthantwo!") if
5 >2:
        print("Fiveisgreaterthantwo!")
```

Youhavetousethesamenumberofspacesinthesameblockofcode,otherwise Python will give you an error:

Example

SyntaxError:

```
if5 >2:
 print("Five is greater than two!")
        print("Fiveisgreaterthantwo!")
```

IDENTIFIERS

IdentifiersinPython

Identifier is a user-defined name given to a variable, function, class, module, etc. The identifier is a combination of character digits and an underscore. They are case-sensitive i.e., ‘num’ and ‘Num’ and ‘NUM’ are three different identifiers in python.It is a good programming practice to give meaningful names to identifiers to makethe code understandable.
Theidentifiernamemustbeunique.

RulesforNamingPythonIdentifiers

- Identifiermustbeunique
- Itcannotbeareservedpythonkeyword.
- Itshouldnotcontainwhitespace.
- ItcanbeacombinationofA-Z,a-z,0-9,orunderscore.
- Itshouldstartwithanalphabetcharacteroranunderscore(_).
- Itshouldnotcontainanyspecialcharacterotherthananunderscore(_).

ExamplesofPythonIdentifiers

Valididentifiers:

- *var1*
- *_var1*
- *_1_var*
- *var_1*

  - myFunc1
    stdent_name
    _check
    ValidNumber

InvalidIdentifiers

- *!var1*
- *1var*
- *1_var*
- *var#1*
- *var1*
- *1number-satrtswithnumber*
- *Studentname–blankspace*
- *Totalamount$-specialcharacter*

BASICSTYLEGUIDELINES

coding and applying logic is the foundation of any programming language butthere’s also another factor that every coder must keep in mind while coding and that is the coding style

Pythonmaintainsastrictwayoforderandformatofscripting.

Sometimes mandatory and is a great help on the user’s end, to understand. Making it easy for others to read code is always a good idea, and adopting a nice coding style helps tremendously for that.

For Python, **PEP 8** has emerged as the style guide that most projects adhere to; it promotes a very readable and eye-pleasing coding style. Every Python developer shouldreaditatsomepoint;herearethemostimportantpointsextractedforyou:

1. Use4-spaceindentationandnotabs.

Examples:

```
#Alignedwithopeningdelimiter.

grow = function_name(variable_one, variable_two,

                     variable_three, variable_four)

# First line contains no argument. Second line onwards

# more indentation included to distinguish this from

#therest.

deffunction_name(

        variable_one, variable_two, variable_three,

        variable_four):

    print(variable_one)
```

The4spaceruleisnotalwaysmandatoryandcanbeoverruledforcontinuationline.

2. Use docstrings : There are both single and multi-line docstrings that can be used in Python. However, the single line comment fits in one line, triple quotes are usedin both cases. These are used to define a particular program or define a particular function.

Example:

```
defexam():

    """Thisissinglelinedocstring"""
```

```
"""Thisis
a
multilinecomment"""
```

3. Wrap lines so that they don't exceed 79 characters : The Python standard library is conservative and requires limiting lines to 79 characters. The lines can be wrapped using parenthesis, brackets, and braces. They should be used in preference to backslashes.

Example:

```
with open('/path/from/where/you/want/to/read/file') as file_one, \
    open('/path/where/you/want/the/file/to/be/written', 'w') as file_two:
    file_two.write(file_one.read())
```

4. Use of regular and updated comments are valuable to both the coders and users : There are also various types and conditions that if followed can be of great help from programs and users point of view. Comments should form complete sentences. If a comment is a full sentence, its first word should be capitalized, unless itisanidentifierthat beginswithalowercaseletter. Inshort comments,the period at the end can be omitted. In block comments, there are more than one paragraphsandeachsentencemustendwithaperiod.Blockcommentsandinlinecomments can be written followed by a single '#'.

Exampleofinlinecomments:

```
geek=geek+1                    #Increment
```

5. Use of trailing commas : This is not mandatory except while making a tuple.

Example:

```
tup=("geek",)
```

5. **Use Python's default *UTF-8* or *ASCII* encodings and not any fancy encodings**, if it is meant for international environment.

6. Use spaces around operators and after commas, but not directly inside bracketing constructs:

```
a=f(1,2)+g(3,4)
```

7. Naming Conventions : There are few naming conventions that should befollowed in order to make the program less complex and more readable. At the same time, the naming conventions in Pythonis a bit of mess, but here are fewconventions that can be followed easily.

There is an overriding principle that follows that the names that are visible to the user as public parts of API should follow conventions that reflect usage rather than implementation.

Herearefewothernamingconventions:
b (single lowercase letter)

B (single upper case letter)

lowercase

lower_case_with_underscores

UPPERCASE

UPPER_CASE_WITH_UNDERSCORES

CapitalizedWords(orCamelCase).ThisisalsosometimesknownasStudlyCaps.

Note:WhileusingabbreviationsinCapWords,capitalizealltheletters

of the abbreviation. Thus HTTPServerError is better than HttpServerError.

mixedCase (differs from CapitalizedWords by initial lowercase character!)

Capitalized_Words_With_Underscores

Inadditiontothesefewleadingortrailingunderscoresarealsoconsidered.

Examples:
single_leading_underscore: weak “internal use” indicator. E.g. from M import * does not import objects whose name starts with an underscore.
single_trailing_underscore_: used to avoid conflicts with Python keyword.

Example:
Tkinter.Toplevel(master,class_='ClassName')

__double_leading_underscore: when naming a class attribute, invokes name mangling.
(inside class FooBar, __boobecomes_FooBar___boo;).
double_leading_and_trailing_underscore: “magic” objects or attributes that live in user-controlled namespaces. E.g. *init*, *importor file*. Only use them as documented.
8. Characters that should not be used for identifiers : ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), or ‘I’ (uppercase letter eye) as single character variable namesas these are similar to the numerals one and zero.
9. Don’t use non-ASCII characters in identifiers if there is only the slightest chance people speaking a different language will read or maintain the code.

10. Nameyourclassesandfunctionsconsistently: Theconventionisto use **CamelCase** for classes and **lower_case_with_underscores** for functions and methods. Always use **self** as the name for the first method argument.

PYTHONOBJECTS

An **Object** is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. Python is an object-oriented programming language that stresses objects

Python Objects are basically an encapsulation of data variables and methods acting on that data into a single entity

Syntax:

```
  obj=MyClass()
print(obj.x)
```

Instance defining represent memory allocation necessary for storing the actual data of variables

CreatingaPythonObject

WorkingoftheProgram: Audi=Cars()

- A block of memory is allocated on the heap. The size of memory allocated is decided by the attributes and methods available in that class(Cars).
- After the memory block is allocated, the special method __init__()iscalled internally. Initial data is stored in the variables through this method.
- The location of the allocated memory address of the instance is returned to the object(Cars).
- Thememorylocationispassedtoself.

```
• class Cars:
•   def __init__(self,m,p):
•     self.model=m
•     self.price=p
•
• Audi=Cars("R8",100000)
```

-
- print(Audi.model)
- print(Audi.price)

Output:

```
R8

100000
```

AccessingClassMemberUsingObject:
Variables and methods of a class are accessible by using class objects or instances in Python.

Syntax:
obj_name.var_name

Audi.model

obj_name.method_name()

Audi.ShowModel();

obj_name.method_name(parameter_list)

Audi.ShowModel(100);

Example1:

```
classCar:#ClassVariable vehicle

   = 'car'



   #Theinitmethodorconstructor

   def __init__(self, model):
      # Instance Variable
        self.model=model
         #Addsaninstancevariable
    def setprice(self, price):
        self.price=price
          #Retrievesinstancevariable def
    getprice(self):
        returnself.price
    # Driver Code
Audi = Car("R8")
Audi.setprice(1000000)
print(Audi.getprice())
```

Output:

```
1000000
```

Example2:

DeletinganObjectinPython:

PythonObjectpropertycanbedeletedbyusingthedelkeyword:

Syntax:

delobj_name.property

objectsalsocanbedeletedbydelkeyword:

Syntax:

delobj_name

STANDARDTYPESANDOTHERBUILT-INTYPES

Thedatastoredinmemorycanbeofmanytypes.

For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Pythonhasfivestandarddatatypes−

- **Numbers**
- **String**
- **List**
- **Tuple**
- **Dictionary**

PythonNumbers

Number data types store numericvalues.Number objects are created when you assign a value to them. For example −

var1=1

var2=10

PythonStrings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

PythonLists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

PythonTuples

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.

PythonDictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Built-inDataTypesinPython

TherearedifferenttypesofdatatypesinPython.Somebuilt-inPythondatatypesare –

- Numericdatatypes−int,float,complex

- Stringdatatypes−str
- Sequencetypes−list,tuple, range
- Binarytypes−bytes,bytearray,memoryview
- Mappingdatatype−dict
- Booleantype−bool
- Setdatatypes−set,frozenset

Python Numeric Data types

InPython,thenumericdatatypeisusedtoholdnumericvalues.

Integers, floating-point, and complex numbers fall under the Python numbers category. They are defined as int, float, and complex classes in Python.

- **int**−holdssignedintegersofnon-limitedlength.
- **float**−holdsfloatingdecimalpoints,andit'saccurateupto15decimalplaces.
- **complex**−holdscomplexnumbers.

Python String Data type

A string is a collection of Unicode symbols. The name for String in Python is str. Single or double quotations are used to represent strings. The use of triple quotes """ or ''' to indicatemultiplestringsisacceptable.Between thequotations,every character is a part of the string.

The only restriction is the machine system's memory resources, which one may use as many characters as they like. In Python programming, deleting or updating a string will result in an error. As a result, the Python programming language does not permit the alteration of strings.

PythonSequenceDatatypes

- **List**− The list is a flexible data type only available in Python. It resembles the arrayinC/C++incertainways.However,thelistinPythonisnoteworthy because it can store many sorts of data simultaneously. A list is an ordered collection of information expressed using commas and square brackets ([]). (,).

- **Tuple**− The list and a tuple are comparable in many respects. Tuples hold a collection of elements of various data kinds, much like lists do. The tuple's components are separated by commas (,) and parenthesized (). Due to the inability to change the elements' size and value, tuples are read-only data structures.
- **Range**− The range() method in Python returns a list of integers that fall inside a specified range. It is most frequently used to iterate over a series of integers using Python loops.

PythonDataBinarytypes

- **bytes**− A bytes object results from the bytes() function. It can produce empty byteobjectsofthedesiredsizeor transformitemsintobyteobjects.Bytes() and bytearray() return different types of objects: bytes() returns an immutable object, whereas bytearray() returns an alterable object.
- **bytearray**− The bytearray object, an array of the specified bytes, is returnedby the bytearray() function. A modifiable series of numbers from 0 to x to 256 is provided.
- **memoryview**− Python programs may access an object's internal data that implements the buffer protocol using memoryview objects without copying. The byte-oriented data of an object may be read and written directly without copying it using the memoryview() method.

PythonMappingDatatype

- **dict**− A**dictionary in Python**is a collection of data items that are stored in an unordered fashion, much like a map. Dictionaries are made up of key-value pairs, as contrast to other data types, which can only contain a single value. Key-value pairs are included in the dictionary to increase its efficiency. A comma "separates each key," whereas each key-value pair in the representation of a dictionary data type is separated by a colon.

PythonBooleanDatatype

- **bool**– True and False are the two pre-built values the boolean type offers. The provided statement's truth or falsity is determined using these values. It's identified by the bool class. Any non-zero integer or the letter "T" can be used to denote truth, while the number "0" or the letter "F" can denote falsehood.

PythonSetDatatypes

- **set**– The data type's unordered collection is called a **Python Set**. It has components that are unique, iterable, and changeable (may change after creation). The order of the items in a set is ambiguous; it can yield theelement's modified sequence. Use the built-in method set() to build the set, or give a list of elements enclosed in curly braces and separated by commas. It may include several kinds of values.
- **frozenset**– The frozenset() method returns an immutable frozenset object whose initial elements are taken from the supplied iterable. A frozen set is an immutable version of a Python set object. The elements of a set can be alteredat any time, but once a frozen set has been created, its elements cannot be altered.

INTERNALTYPES

STANDARDTYPEOPERATORS

Operators in general are used to perform operations on values and variables. These arestandardsymbolsusedforthepurposeoflogicalandarithmeticoperations.In this article, we will look into different types of **Python operators.**

- OPERATORS:Thesearethespecialsymbols.Eg-+,*,/,etc.
- OPERAND:Itisthevalueonwhichtheoperatorisapplied.

TypesofOperatorsinPython

1. ArithmeticOperators
2. ComparisonOperators
3. LogicalOperators
4. BitwiseOperators
5. AssignmentOperators
6. IdentityOperatorsandMembershipOperators

ArithmeticOperatorsinPython

Python Arithmeticoperatorsareusedtoperformbasicmathematicaloperations like **addition, subtraction, multiplication**, and **division**.

Operator	**Description**	**Syntax**
+	Addition: addstwo operands	x+y
–	Subtraction: subtracts two operands	x–y
*	Multiplication: multiplies two operands	x*y
/	Division (float): divides the first operand by the second	x/y
//	Division (floor): divides the first operand by the second	x//y
%	Modulus: returns the remainder when the first operandisdividedbythe	x%y

Operator	Description	Syntax
	second	
**	Power:Returnsfirst	x**y

ComparisonOperatorsinPython

In Python Comparisonof Relational operatorscompares the values. It either returns **True** or **False** according to the condition.

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x>y
<	Less than: True if the left operand is less than the right	x<y
==	Equal to:True if both operands are equal	x==y
!=	Not equal to – True if operandsarenotequal	x!=y
>=	Greater than or equal to True if the left operand is greater than or equal to	x>=y

Operator	Description	Syntax
	theright	
<=	Less than or equal to True if the left operand is less than or equal to the right	x<=y

=isanassignmentoperatorand==comparisonoperator.

PrecedenceofComparisonOperatorsinPython

In python, the comparison operators have lower precedence than the arithmetic operators. All the operators within comparison operators have same precedence order.

ExampleofComparisonOperatorsinPython

Let'sseeanexampleofComparisonOperatorsinPython.

Example: The code compares the values of **'a'**and **'b'**using various comparison operators and prints the results. It checks if **'a'**is greater than, less than, equal to, not equal to, greater than or equal to, and less than or equal to **'b'**.

- Python3

```
a= 13

b=33



print(a > b)

print(a < b)

print(a==b)
print(a  !=  b)

print(a>=b)

print(a<=b)
```

Output

```
False
True
False
True
False
True
```

LogicalOperatorsinPython

Python Logical operatorsperform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to combine conditional statements.

Operator	**Description**	**Syntax**
and	Logical AND: True if both the operands are true	xandy
or	Logical OR: True if either of the operands is true	xory
not	Logical NOT: True ifthe operand is false	notx

PrecedenceofLogicalOperatorsinPython

ThePrecedenceofLogicalOperatorsinpythonisasfollows:

1. Logicalnot
2. logicaland
3. logicalor

ExampleofLogicalOperatorsinPython

The following code shows how to implement Logical Operators in Python:**Example:** The code performs logical operations with Boolean values. It checks if both **‘a'**and **‘b'**are true (**‘and'**), if at least one of them is true (**‘or'**), and negates the value of **‘a'**using **‘not'**. The results are printed accordingly.

- Python3

```
a= True

b = False

print(aandb)

print(a or b)

print(not a)
```

Output

```
False

True

False
```

BitwiseOperatorsinPython

Python Bitwise operatorsact on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	BitwiseAND	x&y
|	BitwiseOR	x| y
~	BitwiseNOT	~x
^	BitwiseXOR	x^y
>>	Bitwiserightshift	x>>
<<	Bitwiseleftshift	x<<

PrecedenceofBitwiseOperatorsinPython

TheprecedenceofBitwiseOperatorsinpythonisasfollows:

1. BitwiseNOT
2. BitwiseShift
3. BitwiseAND
4. BitwiseXOR
5. BitwiseOR

BitwiseOperatorsinPython

HereisanexampleshowinghowBitwiseOperatorsinPythonwork:

Example: Thecodedemonstratesvariousbitwiseoperationswiththevalues of **‘a'**and **‘b'**. It performs bitwise **AND (&)**, **OR (|)**, **NOT (~)**, **XOR (^)**, **right shift (>>)**, and **left shift (<<)** operations and prints the results. These operationsmanipulate the binary representations of the numbers.

- Python3

```
a= 10

b = 4

print(a&b)

print(a | b)

print(~a)

print(a ^ b)

print(a>> 2)

print(a<< 2)
```

Output

```
0
14
-11
14
2
40
```

AssignmentOperatorsinPython

PythonAssignmentoperatorsareusedtoassignvaluestothevariables.

Operator	Description	Syntax

Operator	Description	Syntax
=	Assign the value of the right side of the expression to the left side operand	x=y+z
+=	Add AND: Add right-side operand with left-side operand and then assignto left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and rightoperandsandassign	a%=b a=a%b

Operator	Description	Syntax
	theresulttoleftoperand	
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and	a>>=b a=a>>b

Operator	Description	Syntax
	assign value to left operand	
<<=	Performs Bitwise left shift on operands and assign value to left operand	a<<=b a=a<<b

AssignmentOperatorsinPython

Let’sseeanexampleofAssignmentOperatorsinPython.

Example: The code starts with **‘a'**and **‘b'**both having the value 10. It then performsa series of operations: addition, subtraction, multiplication, and a left shift operation on **‘b'**. The results of each operation are printed, showing the impact of these operations on the value of **‘b'**.

- Python3

```
a= 10

b = a

print(b)

b += a

print(b)

b -= a

print(b)
```

```
b *= a

print(b)

b<<=a

print(b)
```

Output

```
10
20
10
100
102400
```

IdentityOperatorsinPython

In Python, **is** and **is not** are the identity operatorsboth are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

is	Trueiftheoperandsareidentical
isnot	Trueiftheoperandsarenotidentical

ExampleIdentityOperatorsinPython

Let'sseeanexampleofIdentityOperatorsinPython.

Example: The code uses identity operators to compare variables in Python. Itchecks if **'a'**is not the same object as **'b'**(which is true because they have different values) and if **'a'**is the same object as **'c'**(which is true because **'c'**was assignedthe value of **'a'**).

- Python3

```
a= 10

b=20

c= a



print(aisnotb)

print(a is c)
```

Output

```
True
True
```

STANDARDTYPEBUILT-INFUNCTION

Built-infunctionsthatcanbeappliedtoallthebasicobjecttypes: cmp(),repr(), str(), type(), and the single reverse or back quotes (") operator, which is functionally-equivalent to repr().

Function Name	Description
abs()	Returntheabsolutevalueofanumber
aiter()	It takes an asynchronous iterable as an argument and returns an asynchronous iterator for that iterable

Function Name	**Description**
ascii()	Returnsastringcontainingaprintablerepresentationofanobject
bool()	ReturnorconvertavaluetoaBooleanvaluei.e.,TrueorFalse
breakpoint()	It is used for dropping into the debugger at the call site during runtime for debugging purposes
bytearray()	Returnsabytearrayobjectwhichisanarrayofgivenbytes
bytes()	Converts an object to an immutable byte-represented object of a given size and data
chr()	Returns a string representing a character whose Unicode code point is an integer
classmethod()	Returnsaclassmethodforagivenfunction
compile()	ReturnsaPythoncodeobject
eval()	Parses the expression passed to it and runs Python expression(code) within the program
exec()	Usedforthedynamicexecutionoftheprogram

Function Name	Description
max()	
min()	Returns the smallest item in an iterable or the smallest of two or more arguments
pow()	Computethepowerofanumber
round()	Rounds off to the given number of digits and returns the floating-point number
sum()	Sumsupthenumbersinthelist

abs() Function

The python**abs()**function is used to return the absolute value of a number. It takes only one argument, a number whose absolute value is to be returned. The argument can be an integer and floating-point number. If the argument is a complex number, then, abs() returns its magnitude.

Example

1. #integernumber
2. integer=-20
3. **print**('Absolutevalueof-40is:',abs(integer)) 4.
5. #floating number
6. floating=-20.83
7. **print**('Absolutevalueof-40.83is:',abs(floating))

Output:

```
Absolutevalueof-20is:20
Absolutevalueof-20.83is:20.83
```

ascii()Function

The python**ascii()**function returns a string containing a printable representation of an object and escapes the non-ASCII characters in the string using \x, \u or \U escapes.

Example

```
normalText='Pythonisinteresting'
print(ascii(normalText))

otherText='Pythönisinteresting'
print(ascii(otherText))

print('Pyth\xf6nisinteresting')
```

Output:

```
'Python is
interesting''Pyth\xf6nisinterest
ing' Pythön is interesting
```

bool()

The python**bool()**converts a value to boolean(True or False) using the standard truth testing procedure.

Example

```
test1=[]
print(test1,'is',bool(test1))
test1=[0]
print(test1,'is',bool(test1))
test1=0.0
print(test1,'is',bool(test1))
test1= None
```

```
print(test1,'is',bool(test1))
test1=True
print(test1,'is',bool(test1))
test1='Easystring'
print(test1,'is',bool(test1))
```

Output:

```
[]is False
[0]isTrue
0.0 is False
NoneisFalse
True is True
EasystringisTrue
```

eval()Function

Thepython **eval()**functionparsestheexpressionpassedtoitandrunspython expression(code) within the program.

Example

```
x =8
print(eval('x+ 1'))
```

Output:

```
9
```

UNIT-II

NUMBER

Number data types store numeric values. They are immutable data types, which means that changing the value of a number data type results in a newly allocated object.

Therearethreenumerictypesin Python:

- int
- float
- complex

Variablesofnumerictypesarecreatedwhenyouassignavaluetothem:

Example

```
x = 1    # int
y=2.8#float
z= 1j    #complex
```

Int

Int,orinteger,isawholenumber,positiveornegative,withoutdecimals,ofunlimited length.

Example

Integers:

```
x=1
y=35656222554887711
z=-3255522

print(type(x))
print(type(y))
print(type(z))
```

Float

Float,or"floatingpointnumber"isanumber,positiveornegative,containingoneor more decimals.

Example

Floats:

```
x=1.10
y=1.0
z=-35.59

print(type(x))
print(type(y))
print(type(z))
```

Complex

Complexnumbersarewrittenwitha"j"astheimaginarypart:

Example

Complex:

```
x=3+5j
y=5j
z=-5j

print(type(x))
print(type(y))
print(type(z))
```

INTRODUCTIONTONUMBERS

- **Int (signed Integer object) :**they are the negative or non-negative numbers with nodecimalpoint.Thereisnolimitonanintegerinpython.However,thesizeofthe integerisconstrainedbytheamountofthememoryoursystemhas.

Python also provides the support for various number systems like binary, hexadecimal, and octal values. To store values in different number systems, consider the following pattern.

Prefix	Interpretation	Base	Example
0b 0B	Binary	2	0b10 = 2 (decimal) 0b1010=10(decimal)0B101 = 5 (decimal)
0o 0O	Octal	8	0o10 = 8 0o12 = 10 0O132=decimal
0x 0X	Hexadecimal	16	0xA = 10 0xB = 11 0XBE =190

- **float (floating point numbers) :** The float type is used to store the decimal point (floating point) numbers. In python, float may also be written in scientific notation representing the power of 10.forexample,2.5e2representsthevalue250.0.
- **Complex (complex numbers)** Complex numbers are of the form a+bj where a is the real part of the numberand bj is the imaginary part of the number. The imaginary i is nothing but the square root of -1. It is not as much used in the programming.

DOUBLEPRECISIONFLOATINGPOINTNUMBERS

Double-precisionfloating-pointformat(sometimescalled**FP64**or**float64**)is a floating-pointnumber format, usually occupying 64 bitsin computer memory; it representsawidedynamicrangeofnumericvaluesbyusingafloating radixpoint.

Double-precisionbinaryfloating-pointisacommonlyusedformatonPCs,duetoits wider range over single-precision floating point, in spite of its performance and bandwidth cost. It is commonly known simply as *double*.standard specifies a**binary64**ashaving:

- Signbit:1bit
- Exponent:11bits
- Significandprecision:53bits(52explicitlystored)

Thesignbitdeterminesthesignofthenumber(includingwhenthisnumberiszero, which is signed).

Theexponentfieldisan11-bitunsignedintegerfrom0to2047,in biasedform:an exponent value of1023 represents the actualzero. Exponents range from−1022 to +1023becauseexponentsof−1023(all0s)and+1024(all1s)arereservedforspecial numbers.

NUMERICTYPEFUNCTIONS

SEQUENCES:-

Sequencesarecontainerswithitemsstoredin adeterministicordering.Eachsequence data type comes with its unique capabilities.

TherearemanytypesofsequencesinPython. Types of

Sequences

Pythonsequencesareof**sixtypes,**

1. Strings
2. Lists
3. Tuples
4. BytesSequences
5. BytesArrays
6. range()objects

StringsinPython

ThestringisasequenceofUnicodecharacterswritteninsideasingleordouble-quote. Python does not have any **char** type as in other languages (C, C++), therefore, asingle character inside the quotes will be of type **str** only.

1. Todeclarean**emptystring**,use**str()**oritcanbedefinedusingempty stringinside quotes.

ExampleofEmptyStringinPython

name="PythonGeeks"print(n
ame)

Output

```
PythonGeeks
```

2. Stringsare**immutable**datatypes,thereforeoncedeclared,wecan’talterthestring. Though,wecanreassignittoanewstring.

Code

```
name =
"PythonGeeks"print(name[6]
)#outputs'G'name[6]='g'#thro
wserror print(name)
```

Output

```
G
```

ListsinPython

Listsareasinglestorageunittostoremultipledataitemstogether.It’samutable data structure, therefore, once declared, it can still be altered.

Alistcanhold**strings,numbers,lists,tuples,dictionaries,etc.**

1. Todeclarealist,eitheruse**list()**orsquarebrackets[],containingcomma-separated values.

ExampleofListsinPython

```
list_1=["PythonGeeks","Sequences","Tutorial"]#[allstringlist]

print(f'List 1: {list_1}')

list_2=list()#[emptylist]

print(f'List 2: {list_2}')

list_3=[2021,['hello',2020],2.0]#[integer,list,float]

print(f'List 3: {list_3}')

list_4=[{'language':'Python'},(1,2)]#[dictionary,tuple] print(f'List 4:

{list_4}')
```

Output

```
List1:[‘PythonGeeks’,‘Sequences’,‘Tutorial’] List
2: []
List3:[2021,[‘hello’,2020],2.0]
List4:[{‘language’:‘Python’},(1,2)]
```

2. Let’scheckthemutabilityoflists,now.

Code

```
list_1=["PythonGeeks","Sequences","Tutorial"]#[allstringlist]

list_1[2] = "Blog"

print(list_1)
```

Output

```
[‘PythonGeeks’,‘Sequences’,‘Blog’]
```

TuplesinPython

It’sJustlikeLists,Tuplescanstoremultipledataitemsofdifferentdatatypes.The only difference is that they are **immutable** and are stored inside the parenthesis **()**.

1. Todeclareatuple,eitheruse**tuple()**orparenthesis,containingcomma-separated values.

ExampleofTupleinPython:

```
tuple_1=("PythonGeeks","Sequences","Tutorial")#[allstringtuple] print(f'tuple

1: {tuple_1}')

tuple_2=tuple()#[emptytuple]

print(f'tuple 2: {tuple_2}')

tuple_3=[2021,('hello',2020),2.0]#[integer,tuple,float] print(f'tuple

3: {tuple_3}')

tuple_4=[{'language':'Python'},[1,2]]#[dictionary,list] print(f'tuple

4: {tuple_4}')
```

Output:

```
tuple1:(‘PythonGeeks’,‘Sequences’,‘Tutorial’)
tuple 2: ()
tuple3:[2021,(‘hello’,2020),2.0]
tuple4:[{‘language’:‘Python’},[1,2]]
```

2. Let’sesttheimmutabilityoftuplesnow.

Code:

```
tuple_1=("PythonGeeks","Sequences","Tutorial")#[allstringtuple]
```

```
tuple_1[2]="Blog"p

rint(tuple_1)
```

Output:

```
Traceback(mostrecentcalllast):
File“/home/apoorve/Documents/PythonGeeks/python_sequences/main.py”,line2,in
<module>tuple_1[3
]=“Blog”
TypeError:‘tuple’objectdoesnotsupportitemassignment
```

STRINGANDSTRINGOPERATORS:-

A Python String is a sequence of characters. Python Strings are immutable, meaning we can’t modify a string once we declare a string. Python provides a built-in class “str”forhandlingtext,asthetextisthemostcommonformofdataaPythonprogram handles.

- Concatenationoftwoormorestrings.
- Extractingorslicingpartialstringsfromstringvalues.
- Addingorremovingspaces.
- Convertingtolowerorupper case.
- Formattingstringsusingstringformatters.
- Findingand/orreplacingatextinthegivenstringwithanothertext.

Andthelistofoperationsiscountless.Pythonprovidesseveralbuilt-inmethodsthat let us perform operations on a string in a flexible way.

- Assignmentoperator:“=”
- Concatenateoperator:“+”

- Stringrepetitionoperator:“*”
- Stringslicingoperator:“[]”
- Stringcomparisonoperator:“==”&“!=”
- Membershipoperator:“in”&“notin”
- Escapesequenceoperator: “\”
- Stringformattingoperator:“%”& “{}”

ImportanceofStringOperators

StringoperatorsarecrucialinPythonforseveralreasons:

1. **TextManipulation:**Theyenableyoutoperformessentialtextoperationslike concatenation, repetition, and slicing, making it easier to manipulate strings.
2. **DataProcessing:**Stringoperatorsarefundamentalforparsingandprocessing textual data, which is prevalent in many real-world applications.
3. **CodeEfficiency:**Usingstringoperatorsreducestheneedforwritingcomplex code, making programs more concise and readable.
4. **UserInputHandling:**Theyareessentialforvalidatingandprocessinguser input ensuring data integrity in applications.
5. **StringComparison:**Operatorslike==and!=arevitalforcomparingand evaluating strings, allowing for conditional logic in code.
6. **Formatting:**Stringoperatorsaidinformattingtext,makingcreatinguser-friendly outputs and reports easier.

7. **Versatility:**Python’sstringoperatorsprovideflexibility,allowingdevelopers to adapt and manipulate text data as needed for diverse tasks.

String Operators

AssignmentOperator“=”

Python string can be assigned to any variable with an assignment operator “= “. Pythonstringcanbedefinedwitheithersinglequotes[”],doublequotes[“”],ortriple quotes[‘””‘]. var_name = “string” assigns “string” to variable var_name.

Code:

```
string1="hello"
```

```
string2='hello'
```

```
string3='''hello'''
```

```
print(string1)
print(string2)
print(string3)
```

Output: ConcatenateOperator“+”

Twostringscanbeconcatenatedorjoinedusingthe“+”operatorinPython,as explained in the below example code:

Code:

```
string1="hello"
string2="world"
string_combined=string1+string2
print(string_combined)
```

Output:

StringRepetitionOperator“*”

ThesamestringcanberepeatedinPythonbyntimesusingstring*n,asexplainedin the below example.

Code:

```
string1="helloworld"
print(string1*2)
print(string1*3)
print(string1*4)
print(string1*5)
```

Output

Stringslicingoperator“[]”

Characters from a specific string index can be accessed with the string[index] operator.Theindexisinterpretedasapositiveindexstartingfrom0fromtheleftside and a negative index starting from -1 from the right side.

String	**H**	**E**	**L**	**L**	**O**	**W**	**O**	**R**	**L**	**D**
Positive index	0	1	2	3	4	5	6	7	8	9
Negative index	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

- **string[a]:**Returnsacharacterfromapositiveindexaofthestringfromthe left side, as displayed in the index graph above.
- **string[-a]:**Returnsacharacterfromanegativeindexaofthestringfrom the right side, as displayed in the index graph above.
- **string[a:b]:**Returnscharactersfrompositiveindexatopositiveindexbas displayed in the index graph above.
- **string[a:-b]:**Returnscharactersfrompositiveindexatothenegativeindex b of the string as displayed in the index graph above.
- **string[a:]:**Returnscharactersfrompositiveindexatotheendofthestring.

- **string[:b]**Returnscharactersfromthestartofthestringtothepositiveindexb.
- **string[-a:]:**Returnscharactersfromnegativeindexatotheend ofthestring.
- **string[:-b]:**Returnscharactersfromthestartofthestringtothenegativeindex b.
- **string[::-1]:**Returnsastringwithreverseorder.

Code:

```
string1="helloworld"
print(string1[1])
print(string1[-3])
print(string1[1:5])
print(string1[1:-3])
print(string1[2:])
print(string1[:5])
print(string1[:-2])
print(string1[-2:])
print(string1[::-1])
```

Output:

StringComparisonOperator“==”&“!=”

ThestringcomparisonoperatorinPythonisusedtocomparetwostrings.

- The“==”operatorreturnsBooleanTrueiftwostringsarethesame and

  Boolean False if two strings are different.
- The“!=”operatorreturnsBooleanTrueiftwostringsarenotthesameand returns

  Boolean False if two strings are the same.

Theseoperatorsaremainlyusedalongwiththeifconditiontocomparetwostrings where the decision will be taken based on string comparison.

Code:

```
string1="hello"
string2="hello,world"
string3="hello,world"
string4="world"
print(string1==string4)
print(string2==string3)
print(string1!=string4)
print(string2!=string3)
```

Output:

MembershipOperator“in”&“notin”

Themembershipoperatorsearcheswhetherthespecificcharacterispart/memberofa given input Python string.

- **“a”inthestring:**ReturnsbooleanTrueif“a”isinthestringandreturnsFalse if “a” is not in the string.
- **“a”notinthestring:**ReturnsbooleanTrueif“a”isnotinthestringand returns False if “a” is in the string.

Amembershipoperatorisalsousefultofindwhetheraspecificsubstringispartofa given string.

Code:

```
string1="helloworld"
print("w"instring1)
print("W"instring1)
print("t"instring1)
print("t"notinstring1)
print("hello"instring1)
print("Hello"instring1)
print("hello"notinstring1)
```

Output:

EscapeSequenceOperator “\”

Anescapecharacterisusedtoinsertanon-allowedcharacterinthegiveninputstring. An escape character is a “\” or “backslash” operator followed by a non-allowed character.Anexampleofanon-allowedcharacterinaPythonstringisinserting double quotes in the string surrounded by double quotes.

1. Exampleofnon-alloweddoublequotesinPythonstring:

Code:

```
string="HelloworldIamfrom"India""
print(string)
```

Output:

2. Exampleofnon-alloweddoublequoteswith**escapesequence**operator:

Code:

```
string="HelloworldIamfrom\"India\""
print(string)
```

Output:

StringFormattingOperator“%”

The string formatting operator is used to format a string as per requirement. To insert anothertypeofvariablealongwithstring,the“%”operator isusedalongwithPython string. “%” is prefixed to another character, indicating the type of value we want to insertalongwiththePythonstring.Pleaserefertothetablebelowforsomeofthe commonly used different string formatting specifiers:

Operator	Description
%d	Signeddecimalinteger
%u	unsigneddecimalinteger
%c	Character
%s	String
%f	Floating-pointrealnumber

Code:

```
name="india"
age =19
marks=20.56
string1='Hey%s'%(name)
print(string1)
string2='myageis%d'%(age)
print(string2)
```

```
string3='Hey%s,myageis%d'%(name,age)
print(string3)
string3='Hey%s,mysubjectmarkis%f'%(name,marks)
print(string3)
```

Output:

STRINGBUILT-INMETHODS

FunctionName	Description
capitalize()	Converts the first character of the string to a capital (uppercase) letter
casefold()	Implementscaselessstringmatching
center()	Pad the string with the specified character.
count()	Returns the number of occurrences of a substring in the string.
encode()	Encodes strings with the specified encoded scheme
endswith()	Returns “True” if a string ends with the given suffix

FunctionName	Description
expandtabs()	Specifies the amount of space to be substituted with the “\t” symbol in the string
find()	Returns the lowest index of the substring if it is found
format()	Formats the string for printing it to console
format_map()	Formats specified values in a string using a dictionary
index()	Returns the position of the first occurrence of a substring in a string
isalnum()	Checks whether all the characters in a given string is alphanumeric or not
isalpha()	Returns “True” if all characters in the string are alphabets
isdecimal()	Returns true if all characters in a string are decimal
isdigit()	Returns“True”ifallcharactersinthe

FunctionName	Description
	stringaredigits
isidentifier()	Check whether a string is a valid identifier or not
islower()	Checks if all characters in the string are lowercase
isnumeric()	Returns “True” if all characters in the string are numeric characters
isprintable()	Returns “True” if all characters in the string are printable or the string is empty
isspace()	Returns “True” if all characters in the string are whitespace characters
istitle()	Returns “True” if the string is a title cased string
isupper()	Checks if all characters in the string are uppercase
join()	ReturnsaconcatenatedString

FunctionName	Description
ljust()	Left aligns the string according to the width specified
lower()	Converts all uppercase characters in a string into lowercase
lstrip()	Returns the string with leading characters removed
maketrans()	Returnsatranslationtable
partition()	Splits the string at the first occurrence of the separator
replace()	Replaces all occurrences of a substring with another substring
rfind()	Returns the highest index of the substring
rindex()	Returns the highest index of the substring inside the string
rjust()	Right aligns the string according to the width specified

FunctionName	Description
rpartition()	Splitthegivenstringintothreeparts
rsplit()	Split the string from the right by the specified separator
rstrip()	Removestrailingcharacters
splitlines()	Splitthelinesatlineboundaries
startswith()	Returns “True” if a string starts with the given prefix
strip()	Returns the string with both leading and trailing characters
swapcase()	Converts all uppercase characters to lowercase and vice versa
title()	Convertstringtotitlecase
translate()	Modify string according to given translation mappings
upper()	Converts all lowercase characters in a string into uppercase

FunctionName	Description
zfill()	Returns a copy of the string with ‘0’ characters padded to the left side of the string

LIST

Listsareoneof4built-indatatypesinPythonusedtostorecollectionsofdata,the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Listsarecreatedusingsquarebrackets:

Example

CreateaList:

```
thislist=["apple","banana","cherry"]
print(thislist)
```

List Items

Listitemsareordered,changeable,andallowduplicatevalues.

Listitemsareindexed,thefirstitemhasindex[0],theseconditemhasindex [1]etc.

Ordered

Whenwesaythatlistsareordered,itmeansthattheitemshaveadefinedorder,and that order will not change.

Ifyouaddnewitemstoalist,thenewitemswillbeplacedattheendofthelist.

Note: Therearesomelistmethodsthatwillchangetheorder,butingeneral:theorder of the items will not change.

Changeable

Thelistischangeable,meaningthatwecanchange,add,andremoveitemsinalist after it has been created.

AllowDuplicates

Sincelistsareindexed,listscanhaveitemswiththesamevalue:

Example

Listsallowduplicatevalues:

```
thislist=["apple","banana","cherry","apple","cherry"] print(thislist)
```

List Length

Todeterminehowmanyitemsalisthas,usethelen()function:

Example

Printthenumberofitemsinthe list:

```
thislist=["apple","banana","cherry"]
print(len(thislist))
```

ListItems-DataTypes

Listitemscanbeofanydata type:

Example

String,intandbooleandatatypes:

```
list1=["apple","banana","cherry"] list2
= [1, 5, 7, 9, 3]
list3=[True,False,False]
```

Alistcancontaindifferentdatatypes:

Example

Alistwithstrings,integersandbooleanvalues:

```
list1=["abc",34,True,40,"male"]
```

type()

FromPython'sperspective,listsaredefinedasobjectswiththedatatype'list':

```
<class'list'>
```

Example

Whatisthedatatypeofalist?

```
mylist=["apple","banana","cherry"]
print(type(mylist))
```

Thelist()Constructor

Itisalsopossibletousethelist()constructorwhencreatinganewlist.

Example

Usingthelist()constructortomakeaList:

```
thislist=list(("apple","banana","cherry"))#notethedoubleround-brackets print(thislist)
```

LISTTYPEBUILTINMETHODS

S.no	Method	Description
1	append()	Used for adding elements to the end of the List.
2	copy()	It returns a shallow copy of a list
3	clear()	This method is used for removing allitems from the list.
4	count()	These methods count the elements.
5	extend()	Adds each element ofan iterable to the end of the List
6	index()	Returns the lowest index where the element appears.
7	insert()	Inserts a given element at a given index in a list.
8	pop()	Removesandreturnsthe

S.no	Method	Description
		last value from the List or the given index value.
9	remove()	Removes agiven object from the List.
10	reverse()	Reverses objects of the List in place.
11	sort()	Sort aList in ascending, descending, or user-defined order
12	min()	Calculates the minimum of all the elements ofthe List
13	max()	Calculates the maximum of all the elements ofthe List

TUPLES

Tuple is a collection of objects separated by commas. In some ways, a tuple is similar to a Python list in terms of indexing, nested objects, and repetition but the main difference between both is Python tuple is immutable, unlike the Python list which is mutable.

CreatingPythonTuples

There are various ways by which you can create a tuple in Python. They are as follows:

- Usingroundbrackets
- Withoneitem
- TupleConstructor

CreateTuplesusingRoundBrackets()

Tocreateatuplewewilluse()operators.

- Python3

```
var=("Geeks","for","Geeks") print(var)
```

Output:

('Geeks','for','Geeks')

CreateaTupleWithOneItem

Python3.11providesuswithanotherwaytocreateaTuple.

- Python3

```
values:tuple[int|str,...]=(1,2,4,"Geek")

print(values)
```

Output:

Here, in the above snippet we are considering a variable called values which holds a tuple that consists of either int or str, the '…' means that the tuple will hold morethan one int or str.

(1,2,4,'Geek')

TupleConstructorinPython

To create a tuple with a Tuple constructor, we will pass the elements as its parameters.

- Python3

```
tuple_constructor=tuple(("dsa","developement","deeplearning"))

print(tuple_constructor)
```

Output:

```
('dsa','developement','deeplearning')
```

WhatisImmutableinTuples?

Tuples in Python are similar to Python listsbut not entirely. Tuples are immutable andorderedandallowduplicatevalues.SomeCharacteristicsofTuplesinPython.

- We can find items in a tuple since finding any item does not make changes in the tuple.
- Onecannotadditemstoatupleonceitiscreated.
- Tuplescannotbeappendedorextended.
- Wecannotremoveitemsfromatupleonceitiscreated.

```
mytuple=(1,2,3,4,5) #

tuples are indexed

print(mytuple[1])

print(mytuple[4])

 #tuplescontainduplicateelements

mytuple = (1, 2, 3, 4, 2, 3)

print(mytuple)
```

```
 #addinganelement

mytuple[1] = 100

print(mytuple)
```

Output:

Python tuples are ordered and we can access their elements using their index values. They are also immutable, i.e., we cannot add, remove and change the elements once declared in the tuple, so when we tried to add an element at index 1, it generated the error.

```
2
5
(1,2,3,4,2,3)
Traceback(mostrecentcalllast):
  File "e0eaddff843a8695575daec34506f126.py", line 11, in
    tuple1[1] = 100
TypeError:'tuple'objectdoesnotsupportitemassignment
```

AccessingValuesinPythonTuples

TuplesinPythonprovidetwowaysbywhichwecanaccesstheelementsofatuple.

- Usingapositiveindex
- Usinganegativeindex

PythonAccessTupleusingaPositiveIndex

Usingsquarebracketswecangetthevaluesfromtuplesin Python.

```
var = ("Geeks", "for", "Geeks")

print("Value in Var[0] = ", var[0])

print("Value in Var[1] = ", var[1])

print("ValueinVar[2]=",var[2])
```

Output:

```
Value in Var[0] =Geeks
Value in Var[1] =for Value
in Var[2] =Geeks
```

AccessTupleusingNegativeIndex

In the above methods, we use the positive index to access the value in Python, and here we will use the negative index within [].

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

 print("ValueinVar[-1]=",var[-1])

print("ValueinVar[-2]=",var[-2])

print("ValueinVar[-3]=",var[-3])
```

Output:

```
Value   in   Var[-1]=3
Value   in   Var[-2]=2
ValueinVar[-3]=1
```

UNIT-III

MAPPINGTYPE:-

Themappingobjectsareusedtomaphashtablevaluestoarbitraryobjects. In

python there is mapping type called **dictionary**. It is mutable.

Thekeysofthedictionaryarearbitrary.

Asthevalue,wecanusedifferentkindofelementslikelists,integersoranyother mutable type objects.

Somedictionaryrelated**methodsandoperations**are−

Methodlen(d)

Thelen()methodreturnsthenumberofelementsinthedictionary.

Operationd[k]

Itwillreturntheitemofdwiththekey‘k’.Itmayraise **KeyError**ifthekeyisnot mapped.

Methoditer(d)

This method will return an iterator over the keys of dictionary. We can also performthis taks by using **iter(d.keys())**.

M**ethodget(key[,default])**

Theget() method will return thevalue fromthekey.Thesecond argument is optional. If the key is not present, it will return the default value.

Methoditems()

Itwillreturntheitemsusing(key,value)pairsformat.

Methodkeys()

Returnthelistofdifferentkeysinthedictionary.

Methodvalues()

Returnthelistofdifferentvaluesfromthedictionary.

Methodupdate(elem)

Modifytheelementeleminthedictionary.

Examplecodings

```
myDict={'ten':10,'twenty':20,'thirty':30,'forty':40} print(myDict)
print(list(myDict.keys()))
print(list(myDict.values()))

#createitemsfromthekey-valuepairs
print(list(myDict.items()))

myDict.update({'fifty':50})
print(myDict)
```

Output

```
{'ten':10,'twenty':20,'thirty':30,'forty':40}
['ten','twenty','thirty','forty']
[10, 20, 30, 40]
[('ten',10),('twenty',20),('thirty',30),('forty',40)]
{'ten':10,'twenty':20,'thirty':30,'forty':40,'fifty':50}
```

Dictionaries

Dictionariesareusedtostoredatavaluesinkey:valuepairs.

Adictionaryisacollectionwhichisordered*,changeableanddonotallow duplicates.

Dictionariesarewrittenwithcurlybrackets,andhavekeysandvalues: Example

Createandprintadictionary:

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

DictionaryItems

Dictionaryitemsareordered,changeable,anddoesnotallowduplicates.

Dictionaryitemsarepresentedinkey:valuepairs,andcanbereferredtobyusingthe key name.

Example

Printthe"brand"valueofthedictionary:

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

OrderedorUnordered

Dictionariesareordered,itmeansthattheitemshaveadefinedorder,andthatorder will not change.

Unorderedmeansthattheitemsdoesnothaveadefinedorder,youcannotrefertoan item by using an index.

Changeable

Dictionariesarechangeable,meaningthatwecanchange,addorremove itemsafter the dictionary has been created.

DuplicatesNotAllowed

Dictionariescannothavetwoitemswiththesamekey:

Example

Duplicatevalueswilloverwriteexistingvalues:

```
thisdict = {
  "brand":"Ford",
  "model":"Mustang",
  "year": 1964,
  "year":2020
}
print(thisdict)
```

DictionaryLength

Todeterminehowmanyitemsadictionaryhas,usethelen()function:

Example

Printthenumberofitemsinthedictionary:

```
print(len(thisdict))
```

DictionaryItems-DataTypes

Thevaluesindictionaryitemscanbeofanydatatype:

Example

String,int,boolean,andlistdatatypes:

```
thisdict = {
  "brand":"Ford",
  "electric":False,
  "year": 1964,
  "colors":["red","white","blue"]
}
```

type()

FromPython'sperspective,dictionariesaredefinedasobjectswiththedatatype'dict':

<class'dict'>Exa

mple

Printthedatatypeofadictionary:

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

Thedict()Constructor

Itisalsopossibletousethedict()constructortomakea dictionary.

Example

Usingthedict()methodtomakeadictionary:

```
thisdict=dict(name="John",age=36,country="Norway")
print(thisdict)
```

MappingTypeOperators

Dictionaries will work with all of the standard type operators but do not support operationssuchasconcatenationandrepetition.Thoseoperations,althoughtheymake sense for sequence types, do not translate to mapping types. In the next two subsections, we introduce you to the operators you can use with dictionaries.

map() function returns a map object(which is an iterator) of the results after applying the given function to each item of a given iterable (list, tuple etc.)

Python map() Function Syntax

Syntax:*map(fun,iter)*

Parameters:

- ***fun:****Itisafunctiontowhichmappasseseachelementofgiveniterable.*
- ***iter:****Itisiterablewhichistobemapped.*

Map() in Python Examples

Demonstration of map() in Python

Wearedemonstratingthemap()functioninPython.

```
#Pythonprogramtodemonstrateworking #

of map.

 #Returndoubleofn def

addition(n):

    returnn+n
 #Wedoubleallnumbersusingmap()

numbers = (1, 2, 3, 4)

result=map(addition,numbers)

print(list(result))
```

Output

```
[2,4,6,8]
```

map()withLambdaExpressions

We can also use lambda expressionswith map to achieve above result. In this example, we are using map() with lambda expression.

```
#Doubleallnumbersusingmapandlambda numbers =

 (1, 2, 3, 4)

result=map(lambdax:x+x,numbers)

print(list(result))
```

Output

```
[2,4,6,8]
```

AddTwoListsUsingmapandlambda

Weareusingmapandlambdatoaddtwolists.

```
#Addtwolistsusingmapandlambda

 numbers1 = [1, 2, 3]

numbers2=[4,5,6]

 result=map(lambdax,y:x+y,numbers1,numbers2) print(list(result))
```

Output

```
[5,7,9]
```

ModifytheStringusingmap()

We are using map() function to modify the string. We can create a map from an iterable in Python.

```
Listof strings

l=['sat','bat','cat','mat']

 #map()canlistifythelistofstringsindividually test =

list(map(list, l))
```

```
print(test)
```

Output

```
[['s','a','t'],['b','a','t'],['c','a','t'],['m','a','t']]
```

MappingTypeBuilt-inandFactoryFunctions

StandardTypeFunctions[type(),str(),andcmp()]

Thetype()factoryfunction,whenappliedtoadict,returns,asyoumightexpect, the dict type, "<type 'dict'>".

Thestr()factoryfunctionwillproduceaprintablestringrepresentation ofa dictionary. These are fairly straightforward.

we showed how the cmp() BIF worked with numbers, strings, lists, and tuples. So howaboutdictionaries? Comparisonsofdictionariesarebasedonanalgorithmthat starts with sizes first, then keys, and finally values. However, using cmp() on dictionaries isn't usually very useful.

Thefollowingexample,wecreatetwodictionariesandcomparethem,thenslowly modify the dictionaries to show how these changes affect their comparisons:

```
>>>dict1={}>>>dict2={'host':'earth','port':80}
>>>cmp(dict1,dict2)-1
>>>dict1['host']='earth'
>>>cmp(dict1,dict2)-1
```

In the first comparison, dict1 is deemed smaller because dict2 has more elements (2 itemsvs.0items).Afteraddingoneelementto dict1,itisstillsmaller(2vs.1),even if the item added is also in dict2.

```
>>>dict1['port']=8080
 >>>cmp(dict1,dict2)1
>>>dict1['port']=80
 >>>cmp(dict1,dict2)0
```

Afterweaddthesecondelementtodict1,bothdictionarieshavethesamesize, so theirkeysarethencompared.Atthisjuncture,bothsetsofkeysmatch,socomparison proceeds to checking their values. The values for the 'host' keys are the same, but when we get to the 'port' key, dict2 is deemed larger because its value is greater than that of dict1's 'port' key (8080 vs. 80). When resetting dict2's 'port' key to the same value as dict1's 'port' key, then both dictionaries form equals: They have the same size, their keys match, and so do their values, hence the reason that 0 is returned bycmp().

```
>>>dict1['prot']='tcp'
 >>>cmp(dict1,dict2)1
 >>>dict2['prot']='udp'
 >>>cmp(dict1,dict2)-1
```

Assoonasanelementisaddedtooneofthedictionaries,itimmediatelybecomesthe "largerone,"asin thiscasewith dict1.Adding anotherkey-valuepairtodict2 can tip the scales again, as both dictionaries' sizes match and comparison progresses to checking keys and values.

```
>>>cdict={'fruits':1}
 >>>ddict= {'fruits':1}
 >>>cmp(cdict,ddict)0
>>>cdict['oranges']=0
 >>>ddict['apples']=0
>>>cmp(cdict,ddict)14
```

Ourfinalexampleremindsasthat cmp()mayreturnvaluesotherthan -1,0,or1.The algorithm pursues comparisons in the following order.

1)CompareDictionarySizes

Ifthedictionarylengthsaredifferent,then forcmp(dict1,dict2),cmp()willreturna positive number if dict1 is longer and a negative number if dict2 is longer. In other words, the dictionary with more keys is greater, i.e.,

len(dict1)>len(dict2)   dict1> dict2

(2) CompareDictionaryKeys

Ifbothdictionariesarethesamesize,thentheirkeysarecompared;theorderinwhich the keys are checked is the same order as returned by the keys() method. (It is importanttonoteherethatkeysthatarethesamewillmaptothesamelocationsinthe hashtable.Thiskeepskey-checkingconsistent.)Atthepointwherekeysfrombothdo not match, they are directly compared and cmp() will return a positive number if the first differing key for dict1 is greater than the first differing key of dict2.

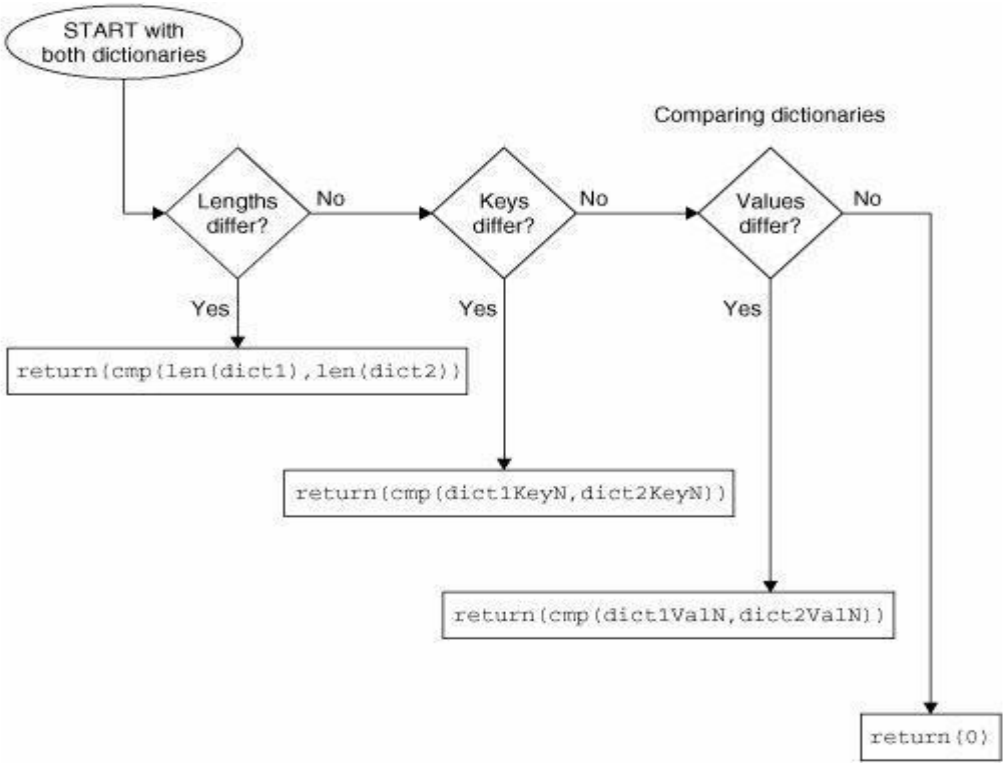
(3) CompareDictionaryValues

Ifbothdictionarylengthsarethesameandthekeysmatchexactly,thevaluesforeach key in both dictionaries are compared.Oncethe first key with non-matching values is found, those values are compared directly. Then cmp() will return a positive number if, using the same key, the value in dict1 is greater than the value in dict2.

(4) ExactMatch

If we have reached this point, i.e., the dictionaries have the same length, the same keys,andthesamevaluesforeachkey,thenthe dictionariesareanexactmatchand0 is returned.

Fig:-Howdictionariesarecompared


START with both dictionaries
Comparing dictionaries
Lengths differ?
No
Keys differ?
No
Values differ?
No
Yes
Yes
Yes
return(cmp(len(dict1),len(dict2))
return(cmp(dict1KeyN,dict2KeyN))
return(cmp(dict1ValN,dict2ValN))
return(0)

MappingTypeRelatedFunctions

dict()

Thedict()factoryfunctionisusedforcreatingdictionaries.

Ifnoargumentisprovided,thenanemptydictionaryiscreated.Thefunhappens when a container object is passed in as an argument to dict().

If the argument is an iterable, i.e., a sequence, an iterator, or an object that supports iteration, then each element of the iterable must come in pairs. For each pair, the first elementwillbeanewkeyinthedictionarywiththeseconditemasitsvalue.Takinga cue from the official Python documentation for dict():

```
>>>dict(zip(('x','y'),(1,2))){'y':2,'x':1}
>>>dict([['x',1],['y',2]]){'y':2,'x':1}
 >>>dict([('xy'[i-1],i)foriinrange(1,3)]){'y':2,'x': 1}
```

Thendict()willjustcreateanewdictionaryandcopythecontentsoftheexistingone. Thenew dictionary isactually ashallow copy oftheoriginaloneand thesameresults can be accomplished by using a dictionary's copy() built-in method. Because creating a new dictionary from an existing one using dict() is measurably slower than usingcopy()

len()

Thelen()BIFisflexible.Itworkswithsequences,mappingtypes,andsets(aswewill find out later on in this chapter). For a dictionary, it returns the total number of items, that is, key-value pairs:

```
>>>dict2={'name':'earth','port': 80}
>>>dict2{'port':80,'name':'earth'}
 >>>len(dict2)2
```

We mentioned earlier that dictionary items are unordered. We can see that above, whenreferencingdict2,theitemsarelistedinreverseorderfromwhichtheywere entered into the dictionary.

hash()

Thehash() BIF is notreally meanttobeused fordictionariesper se,butitcanbeused to determinewhetheran objectis fitto be adictionary key(ornot).Given anobjectas its argument, hash() returns the hash value of that object. The object can only be a dictionary key if it is hashable (meaning this function returns a[n integer] value without errors or raising an exception). Numeric values that are equal hash to thesamevalue(eveniftheirtypesdiffer).A TypeErrorwilloccurifanunhashabletypeis given as the argument to hash()

```
>>> hash([]) Traceback (innermost last):   File"<stdin>",line1,in?TypeError:list
objects are unhashable
```

```
>>>>>>dict2[{}]='foo'Traceback(mostrecentcalllast):           File"<stdin>",line1,in
?TypeError:dictobjectsareunhashable
```

FUNCTION	OPERATIONS
dict([container])	Factoryfunctionforcreatingadictionarypopulatedwithitems from container, if provided; if not, an empty dict is created
len(mapping)	Returnsthelengthofmapping(numberofkey-valuepairs)
hash(obj)	Returnshashvalueofobj

MAPPINGTYPEBUILT-INMETHODS

Python'smap()isabuilt-infunctionthatallowsyoutoprocessandtransform all the items in an iterable without using an explicit for loop, a technique commonly known as mapping. map() is useful when you need to apply a transformation function to each item in an iterable and transform them into a new iterable.

CONDITIONALSANDLOOPS

Conditionalstatementsandloopsarepowerfultoolsforcontrollingtheflowof your program.

Theyallowyoutoexecutespecificblocksofcodebasedoncertainconditionsor to repeat a certain block of code multiple times.

Python provides two basic types of loops to iterate through objects or functions: theforandthewhileloopstatements.Bothlooptypeshaveadditionaloptionsand can be combined with conditional statements.

Conditionalstatementsevaluatebooleanarguments(True/False)usingthe keywords if: ... else: ... .

Conditionalexecution

The if statement

Pythonsupportstheusuallogicalconditionsfrommathematics:

- Equals:a==b
- NotEquals:a!=b
- Lessthan:a<b
- Lessthanorequalto:a<=b
- Greaterthan:a>b
- Greaterthanorequalto:a>=b

Theseconditionscanbeusedinseveralways,mostcommonlyin"ifstatements"and loops.

An"ifstatement"iswrittenbyusingtheifkeyword. genaral

form:

```
ifBOOLEANEXPRESSION:

   STATEMENTS
```

Example:

Ifstatement:

```
a= 33
b=200
ifb>a:
  print("bisgreaterthana")
```

O/P:bisgreaterthana

The if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not.

Syntax:

Else

Theelsekeywordcatchesanythingwhichisn'tcaughtbythepreceding conditions.

Example

```
a=200
b=33
ifb>a:
  print("bisgreaterthana")
elif a == b:
  print("aandbareequal")
else:
  print("aisgreaterthanb")
```

TheelifclauseinPython

Theelifstatementaddsanother"decision"branchto if-else.Let'ssayyouwantto evaluate multiple expressions, then you can use elif as follows:

```
if<expression>:
    <statement(s)>eli
f<expression>:
    <statement(s)>eli
f<expression>:
    <statement(s)>els
 e:
    <statement(s)>
```

Thismeansthatwhentheifstatementisfalse,thenextelifexpressionischecked. When any one expression is true, the control goes outside the if-else block. Atmost,oneblockwouldbeexecuted.Incase elseisnotspecified,andallthe statements are false, none of the blocks would be executed.

Here'sanexample:

```
if51<5:
  print("False,statementskipped")
```

```
elif0<5:
  print("true,blockexecuted") elif
0<3:
  print("true,butblockwillnotexecute") else:
  print("Ifallfails.")
```

Output:

Notethatthesecondelifdidn'texecuteasthefirstelifevaluatedto true.

ConditionalExpressionsinPython

Python's conditional statements carry out various calculations or operations according to whether a particular Boolean condition is evaluated as true or false. In Python, IF statements deal with conditional statements.

We'lllearnhowtouseconditionalstatementsinPythoninthistutorial.

WhatisPythonIfStatement?

To make decisions, utilize the if statement in Python. Ithas abody of instructions that only executes whenever the if statement's condition is met. The additional else statement, which includes some instructions for the else statement, runs if the if condition is false.

Python's if-else statement is used when you wish to satisfy one statement while the other is false.

PythonSyntaxoftheifStatement:

1. **if**<conditionalexpression>
2. Statement
3. **else**
4. Statement

Code

```
#Pythonprogramtoexecuteifstatement

a,b =6,5

#Initializingtheifcondition
ifa>b:
    code="aisgreaterthanb"
    print(code)
```

Output:

```
a is greater than b
```

How to Use the else Condition?

The "else condition" is usually used when judging one statement based on another. If theconditionmentionedintheifcodeblockiswrong,thentheinterpreterwillexecute the else code block.

Code

```
#Pythonprogramtoexecuteif-elsestatement

a,b =6,5

#Initializingtheif-elsecondition
ifa<b:
    code="aislessthan b"
    print(code)
else:
    print("aisgreaterthanb")
```

Output:

```
aisgreaterthanb
```

While statement

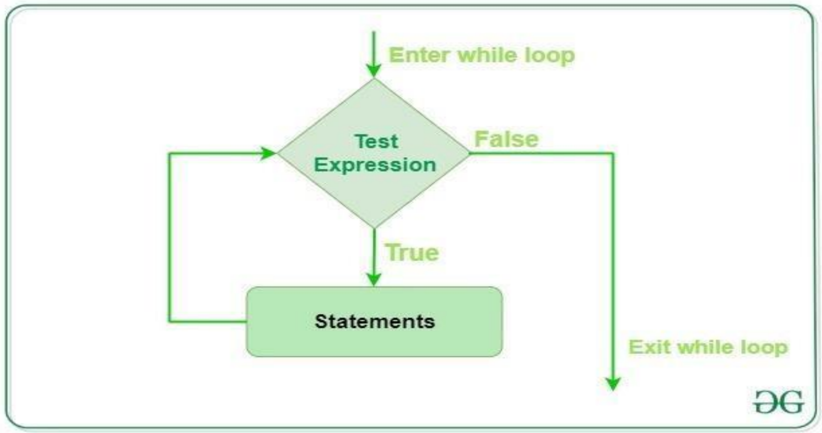
•

PythonWhile Loopis used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:
whileexpression:

    statement(s)

FlowchartofWhilestmt:



Whileloopfallsunderthecategoryof **indefiniteiteration**.Indefiniteiteration means that the number of times the loop is executed isn’t specified explicitly in advance.
Statements represent all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements. When a while loop is executed, expr is first evaluated in a Boolean context and if it is true, the loop body is executed. Then the expr is checked again, if it is still true then the body is executed again and this continues until the expression becomes false.

Example1:PythonWhileLoop

```
#Pythonprogramtoillustrate
```

```
#whileloop

count = 0

while (count < 3):

    count = count + 1

    print("HelloGeek")
```

Output

```
HelloGeek
HelloGeek
HelloGeek
```

In the above example, the condition for while will be True as long as the counter variable (count) is less than 3.

Example2:Pythonwhileloopwithlist

```
#checksiflist still

#containsanyelement a

= [1, 2, 3, 4]

 while a:

    print(a.pop())
```

Output

```
4
3
2
1
```

In the above example, we have run a while loop over a list that will run until there is an element present in the list.

forstatement

Pythonisastrong,universallyapplicableprearranginglanguageplannedtobeeasyto comprehend and carry out. It is allowed to get to because it is open-source. In this tutorial, we will learn how to use Python for loops, one of the most fundamental looping instructions in Python programming.

IntroductiontoforLoopinPython

Python frequently uses the Loop to iterate over iterable objects like lists, tuples, and strings. Crossing is the most common way of emphasizing across a series, for loops are used when a section of code needs to be repeated a certain number of times. The for-circle is typically utilized on an iterable item, for example, a rundown or the in-fabricatedrangecapability.InPython,theforStatementrunsthecodeblockeachtime it traverses a series of elements. On the other hand, the "while" Loop is used when a conditionneedstobeverifiedaftereachrepetitionorwhenapieceofcodeneedstobe repeated indefinitely. The for Statement is opposed to this Loop.

SyntaxofforLoop

```
forvalueinsequence:
    {loopbody}
```

The value is the parameter that determines the element's value within the iterable sequenceoneachiteration.Whenasequencecontainsexpressionstatements,theyare processed first. The first element in the sequence is then assigned to the iterating variable iterating_variable. From that point onward, the planned block is run. Each element in the sequence is assigned to iterating_variable during the statement block until the sequence as a whole is completed. Using indentation, the contents of the Loop are distinguished from the remainder of the program.

ExampleofPythonfor Loop

Code

```
numbers=[3,5,23,6,5,1,2,9,8]
sum_=0
fornuminnumbers:
sum_=sum_ +num** 2
print("Thesumofsquaresis:",sum_)
```

Output:

```
Thesumofsquaresis:774
```

Therange()Function

Since the "range" capability shows up so habitually in for circles, we could erroneously accept the reach as a part of the punctuation of for circle. It's not: It is a built-in Python method that fulfills the requirement of providing a series for the for expression to run over by following a particular pattern . Mainly, they can act straight on sequences, so counting is unnecessary. This is a typical novice construct if they originate from a language with distinct loop syntax:

Code

```
my_list=[3,5,6,8,4]
foriter_varinrange(len(my_list)):
   my_list.append(my_list[iter_var]+2)
print(my_list)
```

Output:

```
[3,5,6,8,4,5,7,8,10,6]
```

BREAKSTATEMENT

Thebreak is a keyword in python which isused to bring theprogram control out of theloop.Thebreakstatementbreakstheloopsone byone,i.e.,inthecaseofnested loops,itbreakstheinnerloopfirstandthenproceedstoouterloops.Inotherwords, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

Thebreakiscommonlyusedinthecaseswhereweneedtobreaktheloopforagiven condition.

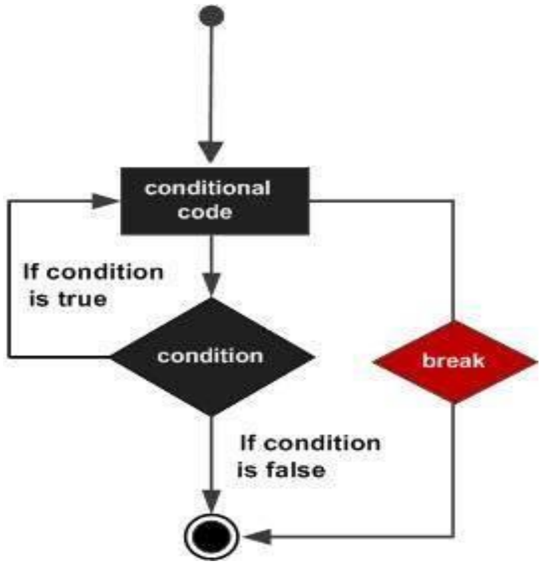
Themostcommonuseforbreakiswhensomeexternalconditionistriggered requiring a hasty exit from a loop. The **break** statement can be used in both*while*and*for*loops.

Ifyouareusingnestedloops,thebreakstatementstopstheexecutionoftheinnermost loop and start executing the next line of code after the block.

Syntax:

1. #loopstatements
2. **break**;

Flow Diagram



Example:breakstatementwithfor loop

Code

```
#breakstatementexample
my_list=[1,2,3,4]
count=1
foriteminmy_list:
    ifitem==4:
        print("Itemmatched")
        count+=1
        break
print("Foundatlocation",count)
```

Output:

```
Item matched
Foundatlocation2
```

CONTINUESTATEMENT

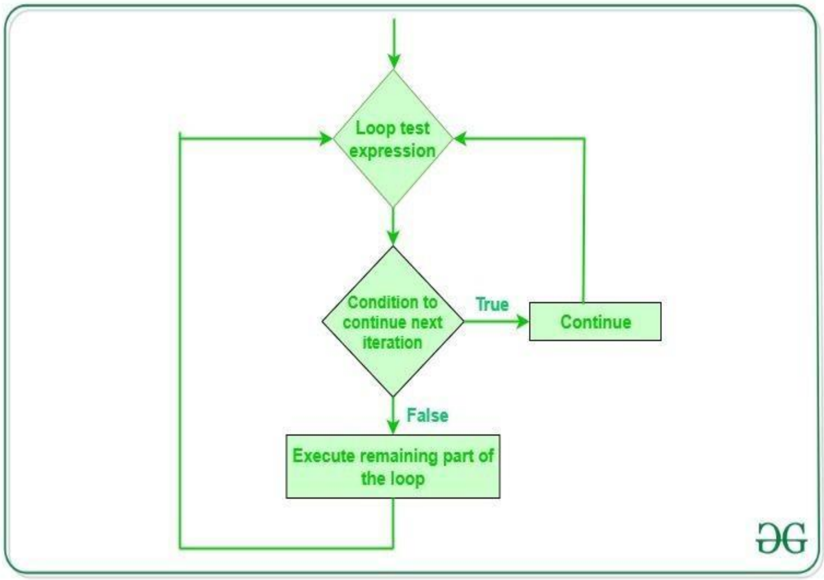
Continue Statement skips the execution of the program block after the continue statement and forces the control to start the next iteration.

Continue statement is a loop control statement that forces to execute the nextiterationoftheloopwhileskippingtherestofthecodeinsidetheloopforthe current iteration only, i.e. when the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped for the current iteration and the next iteration of the loop will begin.

Syntax

```
whileTrue:
    ...
    if x == 10:
        continue
    print(x)
```

FlowchartofContinueStatement


Loop test expression
Condition to continue next iteration
True
Continue
False
Execute remaining part of the loop
ƏG

EXAMPLECODE

```
forvarin"Geeksforgeeks":

    ifvar =="e":

        continue

    print(var)
```

Output:

```
G

k

s

f

o

r

g

k

s
```

Explanation: Here we are skipping the print of character ‘e’ usingif-conditionchecking and continue statement.

PASS STATEMENT

The Pythonpass statement is a null statement. But the difference between pass and commentis that comment is ignored by the interpreter whereas pass is not ignored.

TheSyntaxofthepassstatement

pass

WhatispassstatementinPython?

When the user does not know what code to write, So user simply places a pass at that line. Sometimes, the pass is used when the user doesn’t want any code to execute. So users can simply place a pass where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements. So using a pass statement user avoids this error.

WhyPythonNeeds“pass”Statement?

If we do not use pass or simply enter a comment or a blank here, we will receive an **IndentationError** error message.

```
n=26

 ifn>26:

    #writecodeyourhere

 print('Geeks')
```

Output:

IndentationError:expectedanindentedblockafter'if'statement

keywordinFunction

PythonPasskeywordcanbeusedinemptyfunctions.

```
deffunction():

  pass
```

UseofpasskeywordinPythonClass

The passkeywordcanalsobeusedinanemptyclassinPython.

```
classgeekClass:

  pass
```

UseofpasskeywordinPythonLoop

The pass keyword can be used in Python for loop, when a user doesn’t know what to code inside the loop in Python.

```
n=10

foriin range(n):

     #passcanbeusedas placeholder
```

```
#whencodeistoaddedlater

Pass
```

UseofpasskeywordinConditionalstatement

Pythonpasskeywordcanbeusedwithconditionalstatements

```
a= 10

b=20

 if(a<b):

pass

else:

  print("b<a")
```

Let’s take another example in which the pass statement gets executed when the condition is true.

ITERATORSANDTHEiter()FUNCTION

Aniteratorisanobjectthatcontainsacountablenumberofvalues.

Aniteratorisanobjectthatcanbeiteratedupon,meaningthat youcantraversethrough all the values.

Technically,inPython,aniteratorisanobjectwhichimplementstheiteratorprotocol, which consist of the methods iter() and next().

An iterator in Python is an object that is used to iterate over iterable objects like lists, tuples, dicts, and sets. The Python iterators object is initialized using the**iter()** method.

Itusesthe**next()**methodforiteration.

1. **iter():** The iter() method is called for the initialization of an iterator. This returns an iterator object
2. **next():** The next method returns the next value for the iterable. When we use a for loop to traverse any iterable object, internally it uses the iter() method to get an iterator object, which further uses the next() method to iterate over. This method raises a StopIteration to signal the end of the iteration.

   Pythoniter()Example

```
string = "GFG"

ch_iterator=iter(string)

print(next(ch_iterator))

print(next(ch_iterator))

print(next(ch_iterator))
```

Output:

```
G

F

G
```

IteratorvsIterable

Lists,tuples,dictionaries,andsetsarealliterableobjects.Theyare iterable *containers* which you can get an iterator from.

Alltheseobjectshaveaiter()methodwhichisusedtogetaniterator:

Example

Returnaniteratorfromatuple,andprinteach value:

```
mytuple=("apple","banana","cherry")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

Evenstringsareiterableobjects,andcanreturnaniterator:

Example

Stringsarealsoiterableobjects,containingasequenceofcharacters:

```
mystr                  =
"banana"myit=ite
r(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

LoopingThroughanIterator

Wecanalsouseaforlooptoiteratethroughaniterableobject:

Example

Iteratethevaluesofatuple:

```
mytuple=("apple","banana","cherry")

forxinmytuple:
  print(x)
```

Example

Iteratethecharactersofastring:

```
mystr="banana"

forxinmystr:
  print(x)
```

Theforloopactuallycreatesaniteratorobjectandexecutesthenext()methodforeach loop.

CreateanIterator

Tocreateanobject/classasaniteratoryouhavetoimplementthe methods
iter() and next() to your object.

Allclasseshaveafunctioncalled init(),whichallows youtodosomeinitializing when the object is being created.

Theiter()methodactssimilar, youcandooperations(initializingetc.),butmust always return the iterator object itself.

Thenext() methodalsoallows youtodooperations, and mustreturnthenextitemin the sequence.

Example

Createaniteratorthatreturnsnumbers,startingwith1,andeachsequencewillincrease by one (returning 1,2,3,4,5 etc.):

```
classMyNumbers:
  defiter(self):
    self.a = 1
    returnself

  defnext(self):
    x = self.a
    self.a+=1
    return x

myclass=MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

StopIteration

Theexampleabovewouldcontinueforeverifyouhadenoughnext()statements,orifit was used in a for loop.

Topreventtheiterationfromgoingonforever,wecanusetheStopIterationstatement.

Inthenext() method,wecanaddaterminatingconditiontoraiseanerrorif the iteration is done a specified number of times:

Example

Stopafter20iterations:

```
classMyNumbers:
  defiter(self):
    self.a = 1
    returnself

  defnext(self):
    ifself.a<=20:
      x = self.a
    self.a += 1
    return x
    else:
      raiseStopIteration

myclass=MyNumbers()
myiter = iter(myclass)

forxinmyiter:
  print(x)
```

DefinitionandUsage

Theiter()functionreturnsaniteratorobject.

Syntax

iter(*object*,*sentinel*)

Parameter Values

Parameter	Description
object	Required.Aniterableobject
sentinel	Optional.Iftheobjectisacallableobjectthe iterationwillstopwhenthereturn same as the sentinel

FILESANDINPUT/OUTPUT

Thesimplestwaytoproduceoutputisusingthe *print*statementwhereyoucanpasszero or more expressions separated by commas.

Thisfunctionconvertstheexpressions youpassintoastringandwritestheresultto standard output as follows −

```
#!/usr/bin/python

print"Pythonisreallyagreatlanguage,","isn'tit?"
```

Result:-

```
Pythonisreallyagreatlanguage,isn'tit?
```

Reading Keyboard Input

Pythonprovidestwobuilt-infunctionstoreadalineoftextfromstandardinput,which by default comes from the keyboard. These functions are −

- raw_input
- input

The*raw_input*Function

The*raw_input([prompt])*functionreadsonelinefromstandardinputandreturnsitas a string (removing the trailing newline).

```
#!/usr/bin/python

str=raw_input("Enteryourinput:")
print "Received input is : ", str
```

Thisprompts youtoenteranystringanditwoulddisplaysamestringonthescreen. When I typed "Hello Python!", its output is like this −

```
Enter your input: Hello Python
Receivedinputis:HelloPython
```

The*input*Function

The*input([prompt])*functionisequivalenttoraw_input,exceptthatitassumestheinput is a valid Python expression and returns the evaluated result to you.

```
#!/usr/bin/python

str=input("Enteryourinput:") print
"Received input is : ", str
```

Thiswouldproducethefollowingresultagainsttheenteredinput− Enter

```
your input: [x*5 for x in range(2,10,2)]
Recievedinputis:[10,20,30,40]
```

OpeningandClosingFiles

ReadingandWritingtothestandardinputandoutput.Now,wewillseehowtouseactual data files.

Pythonprovidesbasicfunctionsandmethodsnecessarytomanipulatefilesbydefault. You can do most of the file manipulation using a **file** object.

The*open*Function

Beforeyoucanreadorwriteafile, youhavetoopenitusingPython'sbuilt-in*open()*function.Thisfunctioncreatesa**file**object,whichwouldbeutilizedtocall other support methods associated with it.

Syntax

fileobject=open(file_name[,access_mode][,buffering])

Hereareparameterdetails−

- **file_name**−Thefile_nameargumentisastringvaluethatcontainsthenameof the file that you want to access.
- **access_mode**−Theaccess_modedeterminesthemodeinwhich thefilehastobe opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering** − If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performedwiththeindicatedbuffersize.Ifnegative,thebuffersize isthesystem default(default behavior).

Sr.No.	Modes&Description
1	**R** Opensafileforreadingonly.Thefilepointerisplacedatthebeginning of the file. This is the default mode.
2	**Rb** Opensafileforreadingonlyinbinaryformat.Thefilepointerisplaced at the beginning of the file. This is the default mode.
3	**r+** Opensafileforbothreadingandwriting.Thefilepointerplacedatthe beginning of the file.
4	**rb+** Opensafileforbothreadingandwritinginbinaryformat.Thefile pointer placed at the beginning of the file.
5	**W** Opensafileforwritingonly.Overwritesthefileifthefileexists.Ifthe filedoesnotexist,createsanewfileforwriting.

6	**Wb** Opensafileforwritingonlyinbinaryformat.Overwritesthefileifthe file exists. If the file does not exist, creates a new file for writing.
7	**w+** Opensafileforbothwritingandreading.Overwritestheexistingfileif thefileexists.Ifthefiledoesnotexist,createsanewfileforreading and writing.

The*file*ObjectAttributes

Onceafileisopenedand youhaveone*file*object, youcangetvariousinformation related to that file.

Sr.No.	Attribute&Description
1	**file.closed** Returnstrueiffileisclosed,falseotherwise.
2	**file.mode** Returnsaccessmodewithwhichfilewas opened.
3	**file.name** Returnsnameofthefile.
4	**file.softspace** Returnsfalseifspaceexplicitlyrequiredwithprint,trueotherwise.

Example

```
#!/usr/bin/python

# Open a file
fo=open("foo.txt","wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print"Softspaceflag:",fo.softspace
```

Thisproducesthefollowingresult− Name

```
of the file:foo.txt
Closedornot:False Opening
mode :wb Softspace flag :0
```

The*close()*Method

Theclose()methodofa*file*objectflushesanyunwritteninformationandclosesthefile object, after which no more writing can be done.

Pythonautomaticallyclosesafilewhenthereferenceobjectof afileisreassignedto another file. It is a good practice to use the close() method to close a file.

Syntax

```
fileObject.close()
```

Example

```
#!/usr/bin/python

# Open a file
fo=open("foo.txt","wb")
print"Nameofthefile:",fo.name

#Closeopendfile
fo.close()
```

Thisproducesthefollowingresult−

Nameofthefile:foo.txt

ReadingandWritingFiles

The*file*objectprovidesasetofaccessmethodstomakeourliveseasier.Wewouldsee how to use *read()* and *write()* methods to read and write files.

The*write()*Method

The*write()*methodwritesanystringtoanopenfile.ItisimportanttonotethatPython strings can have binary data and not just text.

Thewrite()methoddoesnotaddanewlinecharacter('\n')totheendofthestring−

Syntax

```
fileObject.write(string)
```

Example

```
#!/usr/bin/python

#Opena file
fo=open("foo.txt","wb")
fo.write("Pythonisagreatlanguage.\nYeahitsgreat!!\n")

#Closeopendfile
fo.close()
```

Theabovemethodwouldcreate*foo.txt*fileandwouldwritegivencontentinthatfileand finally it would close that file. If you would open this file, it would have following content.

```
Pythonisagreatlanguage. Yeah
its great!!
```

The*read()*Method

The*read()*methodreadsastringfromanopenfile.ItisimportanttonotethatPython strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count])
```

Here, passed parameter is the number of bytes to be read from the opened file. This methodstartsreadingfromthebeginningofthefileandif *count*ismissing,thenittriesto read as much as possible, maybe until the end of file.

Example

```
#!/usr/bin/python

#Opena file
fo=open("foo.txt","r+") str
= fo.read(10);
print"ReadStringis:",str #
Close opend file fo.close()
```

Thisproducesthefollowingresult−

```
Read String is :Python is
```

FilePositions

The*tell()*methodtellsyouthecurrentpositionwithinthefile;inotherwords,thenext read or write will occur at that many bytes from the beginning of the file.

The*seek(offset[,from])*methodchangesthecurrentfileposition.The*offset*argument indicatesthenumberofbytestobe moved.The*from*argumentspecifiesthereference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 meansusethecurrentpositionasthereference positionandifitissetto2thentheendof the file would be taken as the reference position.

Example

```
#!/usr/bin/python

#Opena file
fo=open("foo.txt","r+") str
= fo.read(10)
print"ReadStringis:",str

#Checkcurrentposition
position = fo.tell()
print"Currentfileposition:",position
```

```
#Repositionpointeratthebeginningonceagain position
= fo.seek(0, 0);
str=fo.read(10)
print"AgainreadStringis:",str #
Close opend file
fo.close()
```

Thisproducesthefollowingresult−

```
Read String is :Python is
Current file position
:10AgainreadStringis:Pythonis
```

RenamingandDeletingFiles

Python**os**moduleprovidesmethodsthathelpyouperformfile-processingoperations, such as renaming and deleting files.

Tousethis module youneedtoimportitfirstandthen youcancallanyrelatedfunctions. The

rename() Method

The*rename()*methodtakestwoarguments,thecurrentfilenameandthenewfilename.

Syntax

```
os.rename(current_file_name,new_file_name)
```

Example

Followingistheexampletorenameanexistingfile*test1.txt*−

```
#!/usr/bin/python
import os

#Renameafilefromtest1.txttotest2.txt
os.rename( "test1.txt", "test2.txt" )
```

The*remove()*Method

Youcanusethe*remove()*methodtodeletefilesbysupplyingthename of thefiletobe deleted as the argument.

Syntax

os.remove(file_name)

Example

Followingistheexampletodeleteanexistingfile*test2.txt*−

```
#!/usr/bin/python
importos

# Delete file test2.txt
os.remove("text2.txt")
```

FILEBUILT-INFUNCTIONS

[open()]

A built-in function open() to open a file. Which accepts two arguments, file name and access mode in which the file is accessed. The function returns a fileobject which can be used to perform various operations like reading, writing, etc.

Syntax:

Fileobject=open(file-name,access-mode)

file-name:Itspecifiesthenameofthefiletobeopened.

access-mode: Therearefollowingaccessmodesforopeningafile:

access-mode	**Description**
"w"	Write-Opensafileforwriting,createsthefileifitdoesnotexist.

access-mode	Description
"r"	Read-Defaultvalue.Opensafileforreading,errorifthefiledoesnotexist.
"a"	Append-Opensafileforappending,createsthefileifitdoesnotexist.
"x"	Create-Createsthespecifiedfile,returnsanerroriffthefileexists.
"w+"	Openafileforupdating(readingandwriting),overwriteifthefileexists.
"r+"	Openafileforupdating(readingandwriting),doesn’toverwriteifthefileexists.

thefileshouldbehandledasbinaryortext mode

access-mode	Description
"t"	Text-Defaultvalue.Textmode.
"b"	-Binary-Binarymode(e.g.images)

EXAMPLE

```
fileptr=open("myfile.txt","r") if
fileptr:
   print("fileisopenedsuccessfullywithreadmodeonly")
```

Example:

```
fileptr=open("myfile1.txt","x") if
fileptr:
   print("newfilewascreatedsuccessfully")
```

Output:

```
fileisopenedsuccessfullywithreadmodeonly
```

FILEBUILT-INMETHODS:-

Filehandlingandallowsuserstohandlefilesi.e.,toreadandwritefiles,alongwith many other file handling options, to operate on files.

Built–infunctions,

- **close()**
- **read()**
- **readline()**
- **write()**
- **writelines()**
- **tell()**
- **seek()**

close()

The close() method used to close the currently opened file, after which no more writingor Reading can be done. Automatically closes a file when the reference object of a file is reassigned to anotherfile. It is a good practice to use the close() method to close a file.

Syntax:

Fileobject.close()

Example:

```
f=open("myfile.txt","r") f.close()
```

read()

Theread()methodisusedtoreadthecontentfromfile.Toreadafilein Python,we must open the file in reading mode.

Syntax:

Fileobject.read([size])

Where‘size’specifiesnumberofbytestoberead.

myfile.txt

```
functionopen()toopenafile.
method read() to read a file.
```

Example:

```
#Openingafilewithreadmode
fileptr = open("myfile.txt","r")
if fileptr:
            print("fileisopenedsuccessfully")
            content=fileptr.read(5)#read5characters
            print(content)
            content=fileptr.read()#readallcharacters
            print(content)
else:
            print("filenotopened")
fileptr.close();
```

Output:

```
fileisopenedsuccessfully funct
ion open() to open a file.
methodread()toreadafile.
```

readline()

To read the file line by line by using a function readline(). The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Syntax:

Fileobject.readline()

myfile.txt

```
functionopen()toopenafile.
method read() to read a file.
```

Example:

```
fileptr=open("myfile.txt","r")
if fileptr:
            print("fileisopenedsuccessfully")
            content=fileptr.readline()
            print(content)
            content=fileptr.readline()
            print(content)
fileptr.close();
```

Output:

```
file is opened successfully
functionopen()toopenafile.
method read() to read a file.
```

write()

The write () method is used to write the content into file. To write some text to a file, we need to open the file using the open method with one of the following access modes.

w: It will overwrite the file if anyfile exists. The file pointer point at the beginning of the file in this mode.

Syntax:

Fileobject.write(content)

```
functionopen()toopenafile.
method read() to read a file.
```

Example:

```
fileptr = open("myfile.txt","w");
#appendingthecontenttothefile
fileptr.write("Pythonisthemoderndaylanguage.")
#closing the opened file
fileptr.close();
```

myfile.txt:

```
Pythonisthemoderndaylanguage.
```

a:Itwillappendtheexistingfile.Thefilepointerpointattheendofthefile.

myfile.txt

```
functionopen()toopenafile.
method read() to read a file.
```

Example:

```
fileptr = open("myfile.txt","a");
#appendingthecontenttothefile
fileptr.write("Pythonisthemoderndaylanguage.")
#closing the opened file
fileptr.close();
```

myfile.txt:

```
functionopen()toopenafile.
method read() to read a file.
Pythonisthemoderndaylanguage.
```

Now,wecanseethatthecontentofthefileis modified.

writelines()

Thewritelines() methodisusedtowrite multiplelinesof contentintofile.Towritesome lines to a file

Syntax:

Fileobject.writelines(list)

list−ThisistheSequenceofthestrings.

Example:

```
f=open("myfile.txt","w")
f.writelines(["PythonsupportsFiles\n","pythonsupportsStrings."]) f.close()
```

myfile.txt:

```
Python supports Files
pythonsupportsStrings.
```

tell()

Thetell()methodreturnsthecurrentfilepositioninafilestream.Youcanchangethe current file position with the seek() method.

Syntax:

Fileobject.tell()

myfile.txt

```
functionopen()toopenafile.
method read() to read a file.
```

Example:

```
f=open("myfile.txt","r")
print(f.readline())
print(f.tell())
f.close();
```

Output:

```
33
```

Inthefistlineoffilethatis"functionopen()toopenafile."with32charecterssothe outputis 33

seek()

Theseek()methodsetsandreturnsthecurrentfilepositioninafilestream.

Syntax:

Fileobject.seek(offset)

myfile.txt

```
functionopen()toopenafile.
method read() to read a file.
```

Example:

```
f=open("myfile.txt","r")
print(f.seek(9))
print(f.read())
f.close();
```

Output:

```
open() to open a file.
methodread()toreadafile.
```

FILEBUILT-INATTRIBUTES

PythonSupportsfollowingbuilt-inattributes,thoseare

- **file.name**-returnsthenameofthefilewhichisalreadyopened.
- **file.mode**-returnstheaccessmodeofopenedfile.

- **file.closed**-returnstrue,ifthefileclosed,otherwisefalse.

Example:

```
f=open("myfile.txt","r")
print(f.name)print(f.mode)
print(f.closed)
f.close()
print(f.closed)
```

Output:

```
myfile.txt
r
False
True
```

Pythonprogram to print number of lines, words and characters in given file.

myfile.txt

```
functionopen()toopenafile.
method read() to read a file.
Pythonisthemoderndaylanguage.
Python supports Files
pythonsupportsStrings
```

Example:

```
fname=input("Enterfilename:") num_lines
= 0
num_words=0
num_chars=0 try:
        fp=open(fname,"r")
        for i in fp:
                #icontainseachlineofthefile words =
                i.split()
                num_lines += 1
                num_words+=len(words)
                num_chars += len(i)
        print("Lines=",num_lines)
```

```
            print("Words = ",num_words)
            print("Characters=",num_chars)
 fp.close()
exceptException:
            print("Entervalidfilename")
```

Output:Case1

```
Enterfilename:myfile.txt Lines
=5
Words=24
Characters=144
```

Output:Case2

```
Enter file name: gh
Entervalidfilename
```

COMMAND-LINEARGUMENTSINPYTHON

Python using raw_input() or input(). There is another method that uses command line arguments. The command line arguments must be given whenever we want to give the inputbeforethestartofthescript,whileontheotherhand,input()isusedtogettheinput while thepython program / script is running

How touseit?

Touseit, youwillfirsthavetoimportit(import sys)

Thefirstargument,sys.argv[0],isalwaysthenameoftheprogramasitwasinvoked, and sys.argv[1] is the first argument you pass to the program.

It'scommonthat youslicethelisttoaccesstheactualcommandlineargument:

Thesysmodulealsoprovidesaccesstoanycommand-lineargumentsviasys.argv. Command-line

argumentsarethoseargumentsgiventotheprograminadditiontothescriptnameon invocation.

- sys.argvisthelistofcommand-linearguments
- len(sys.argv)isthenumberofcommand-linearguments.

.**Example:**

```
#filename"cmdarg.py"impo
rt sys
program_name=sys.argv[0]
arguments = sys.argv[1:]
count = len(arguments)
print(program_name)
print(arguments)
print("Numberofarguments",count)
```

Output:

```
pythoncmdarg.py4556
cmdarg.py
['45','56']
Numberofarguments2
```

UNIT –4

Functions and Functional Programming – Functions – calling functions – creating functions – passing functions – Built-in Functions: apply( ), filter( ), map( ) and reduce( ) - Modules – Modules and Files – Modules built-in functions - classes – class attributes – Instances.

FunctionalProgramming

Functionalprogramming isdesignedto handlethesymboliccomputationandapplicationprocessing list, and it is based on mathematicalwork. The most popular functionalprogramming languages are Python, Lisp, Haskell, Clojure, Erlang etc.

FunctionalProgramminghastwotypes;thoseareshownasbelow:

Pure Functional Languages: Pure functional language supports only the functional pattern. An example of the pure functional language is Haskell.

Impure Functional Language: Impure Functional language supports the prototype of functions and the programming's imperative style. An example of an impure functional language is LISP.

CharacteristicsoftheFunctionalProgrammingLanguage

Characteristicsofthefunctionalprogramminglanguagesarelikeasshownbelow:

Functional programming languages are designed to perform the functions of mathematical functions. These functions use conditional expressions and recursion to perform the computation.

Functionalprogrammingsupportsfunctions in**higher-order**andfeaturesof**lazyevaluation**.

FunctionalProgramming language directlyusesthe functionsand functioncalls. It doesnot supportthe flow ofthe controls like statements ofthe loop, and statements are like the conditional statements such as If-Else and Switch Statements.

Object-Oriented Programming supports the Abstraction, Encapsulation, and Polymorphism, just like functional programming languages support OOPS concepts.

Advantagesofthe FunctionalProgramming

Advantagesofthefunctionalprogramminglanguagesareasshownbelow:

Bugs-Free code: FunctionalProgramming language doesnot support **state**, so there isno side effect of the functional programming; hence we can write the error-free code.

Efficient Programming Language: Functional Programming language has no mutable state, so thereisno statechange issue. We candothe program"Functions" to workparallelto "Instruction". Thistype of code supports reusability and testability easily.

Efficiency-Functional Program containstheIndependentUnits.Independentunitsrun concurrently. Hence these functional programs are more efficient.

SupportsNestedFunctions-Nestedfunctionsaresupportedbyfunctionalprogramming.

Lazy Evaluation- Lazy Functional Constructions are also supported by functional programming such as Lazy Lists, Lazy Maps, etc.

Functionalprogrammingdoesnothaveanystate,so allthetime, there isa needto createnewobjectsto perform the actions. That's why functional programming needs a large memory space.

Functionalprogrammingisused to performthedifferent operationsonthesamedataset.

The LISP supports artificial intelligence applications such as language processing, Machine learning,Modelling of speech and vision.

DifferencesbetweentheFunctionalProgrammingandObject-OrientedProgrammingare:

Sr.No.	FunctionalProgramming	Object-OrientedProgramming
1.	Thefunctionalprogramminglanguage supports immutable data.	OOPusesmutable data.
2.	FunctionalProgrammingsupportsthe Declarative Programming Model.	OOPsupportstheimperativeProgramming Model.
3.	Functional Programming focuses on the "What we are doing".	OOPfocusesonthe "Howwearedoing".
4.	The methods of Functional Programmingwill not produceanyside-effects.	MethodsoftheOOPcanproducetheside-effects.
5.	Functional Programming follows parallel programming.	OOPdoesnotworkonparallelprogramming.
6.	Fortheflowcontrol,wedofunction calls & function calls with recursion.	Object-Oriented Programming supports the use of the loops and conditional statements for the flow control.
7.	For the iteration of the data collection, functional programming uses the"Recursion" concept.	Object-Oriented Programming uses the "Loop" concept for the iteration of Data collection. For example, For-each loop in Java
8.	For functional programming, the execution of statements in the order is not so important.	It is essential for the oop programming to execute the statements in order is very important.
9.	Functional Programming supports "Abstraction over Data" and "Abstraction over Behavior".	OOPsupportsonly"AbstractionoverData".

EfficiencyofProgram

The program'scode isdirectlyproportionaltothe efficiencyofthe algorithmand the executionspeedof the program. Ifthe efficiency is good, that means the performance will be high in the program.

Theefficiencyoftheprogramisaffected bythe belowfactors:

Themachine'sspeed affectstheefficiencyofthe program.

Thecompiler'sspeedalsoaffectstheefficiencyofthe program.

The operating systemalso plays a crucialrole inthe efficiencyofthe programming code.

Thechoiceoftheright Programming languageaffectstheefficiencyoftheprogramming. Data

organization is also affecting the efficiency of the program.

The use of the algorithm in the program affects the efficiency of the programs. An algorithm in the Functional Programming solves the problem.

Wecan increasetheefficiencyofthe programming languagewiththehelp ofthebelowtasks-

Toincreasetheprogram'sefficiency, we have to removetheprogram'sunusablecodeorthecodethat is having redundant processing.

Theuseofoptimalmemoryandnon-volatilestoragehelpstoimprovetheefficiencyofthe programming language.

Wecanreusethecomponents.Thiswillalso helptoincreasetheefficiencyoftheprogram. By using

the error & exception handling on all the layers of the program.

Duringtheprogram'scoding,theprogramshouldhavetobeensuredaboutthedata'sintegrityand consistency.

Byusingtheprogramming code,wecando designlogicandflow.

Efficientprogrammingcodecanreducetheconsumptionoftheresourcesandtimetakenbythecompletion programs.

FunctionalProgramming-CallbyValue

After defining the function, there is a need to pass the arguments into the function to get the desired output. All the programming language supports the **call by value**and**call by reference** methods after arguments passed into the functions.

"Callbyvalue"worksonlyontheobject-orientedprogramminglanguagelikeC++.Functional programming language like Python also supports "call by value".

The original value of the call by value method will not change when we pass the arguments to the function. The function will store the value locally through the function parameter in stack's memory. The changed value in the function will not affect functions from the outside.

CallbyValue inC++

Hereisthe programwhichshowsthecallby valueinthe C++.

```
#include<iostream.h>
#include<conio.h>

voidswap(intx,inty)
{
 inttemp;
 temp=x;
 x=y;
 y=temp;
}

voidmain()
{
 intx=400,y=600;
 clrscr();
 swap(x, y);//argumentspassedtothefunction
 cout<<"Value of x"<<x;
 cout<<"Valueofy"<<y;
 getch();
}
```

Thiswillshowthefollowing output:

```
Valueofxis:200Val
ueofyis:100
```

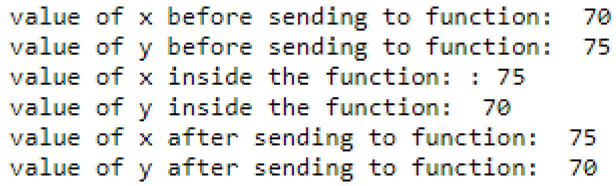
CallbyValue inPython

Belowprogramshowstheworking ofCallby ValueinPython:

```
defswap(x,y): t
= x;
x=y;
```

```
y=t;
print  "thevalueofxargument  insidethe  function:  ",x
print "thevalueofyargument insidethe function: ",y #
Now we will call swap function
x=70
y=75
print "thevalueofxargument beforeassigning intofunction:",x print
"thevalueofyargument beforeassigning intofunction:", y swap (x,
y)
print  "thevalueofxargument  afterassigning  intofunction:",y
print"thevalueofyargumentafterassigningintofunction:",x
```

Theoutputoftheaboveprogramwilllooklikeasshownbelow:



```
value of x before sending to function:  70
value of y before sending to function:  75
value of x inside the function: : 75
value of y inside the function:  70
value of x after sending to function:  75
value of y after sending to function:  70
```

Function Overloading

When any program contains multiple functions, whose name is the same, but the parameters are different, they are known as overloaded. We use this technique to increase the readability of the program.

Tooverloadthefunctions,wewillusetwomethods. - The

function contains a different number ofarguments.

Function having different types of arguments.

Weusuallywilldothefunctionoverloadingwhenwewanttoperformoneoperation,containing differentnumbersortypesofarguments.

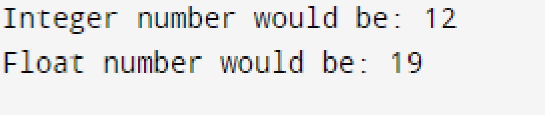
FunctionOverloadinginC++

FunctionOverloadinginC++willbelikeasshownbelow: #

```
include <iostream>
usingnamespacestd;
void addNum(int, int);
voidaddNum(int,int,int);
int main()
{
```

```
addNum(5,7);
addNum(5,6,8);
return0;
}
voidaddNum(inta,intb)
{
cout<<"Integernumberwouldbe: "<<a+b<<endl;
}
voidaddNum(int a,int b, intc)
{
cout<<"Floatnumberwouldbe: "<<a+b+c<<endl;
}
```

Theoutputoftheabove programwilllooklike,asshownbelow:



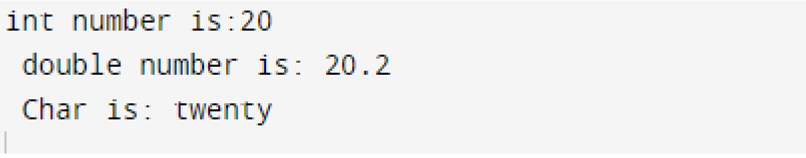
```
Integer number would be: 12
Float number would be: 19
```

Nowwewilltakeanotherexampleofthefunction overloading.

```
#include<iostream>
usingnamespacestd;

voidprint(intg)
{
cout<<"intnumberis:"<<g<<endl;
}
voidprint(doublef)
{
cout<<"doublenumberis:"<<f<<endl;
}
void print(charconst*c)
{
cout<<"Charis:"<<c<<endl;
}

intmain()
{
print(20);
print(20.20);
print("twenty");
return0;
}
```

Nowtheoutput oftheaboveprogramwilllooklikeasshowninthebelow screenshot:

Output



```
int number is:20
 double number is: 20.2
 Char is: twenty
```

FunctionOverloadinginErlang

In the system, the Overloading process regulates the use of the CPU. In the function overloading, the main application calls the request function before doing any job and executes the process when it returns the positive value; else, the job will not start.

Overload is the part of the sas1 application, and we can define all the configuration parameter here.

We will maintain the two sets of intensity; those are the **total intensity** and the **accept intensity**. Intensitiescanbemeasuredthroughtheconfigurationparameters,whichare: **MaxIntensity** and the **Weight value**. Both the intensities will be measured according to the 1/second.

Totalintensitycan becalculated,asshownbelow:

The assumption is thatthe current callto request/0is K(n), and thetime ofthe previous callwas K(n-1). The current total intensity is denoted as KI(n). We will calculate the intensity through the below formulas:

KI(m)=exp(-Weight*(K(m)-K(m-1))*KI(m-1)+Weight

WhereKI(n-1)istheprevioustotalintensity.

TheacceptintensityisdenotedasBI(n),currentacceptintensitycanbedefinedasshown below:

BI(n)=exp(-Weight*(T(m)-T(m-1))*AI(m-1)+Weight

where AI(n-1) isknownasthepreviousaccept intensity, providedthatthevalueofexp(-Weight*(T(n) - T(n-1)) * AI(n-1) is less than MaxIntensity; else the value is

AI(n)=exp(-Weight*(T(n)- T(n-1))*AI(n-1)

Speed is controlled by the value of the configuration parameter (Weight), and the intensities' calculationswillreact accordingtothechanges inthe input intensity. The invertedvalueofWeight will be denoted like as shown below,

T=1/Weight

This value can be defined as the "time constant," which is the intensity calculation formulas. For example, if Weight = 0.1, then the input intensity change is denoted by the total and accepts the 10 seconds' intensities. The overload process defined one alarm, which sets the alarm_handler:set_alarm(Alarm). We will define the alarm as:

{overload,[]}

Wewillset thisalarmwhenthecurrentacceptintensityexceeds**MaxIntensity**.

Nowwewillperformfunctionoverloadingin Erlang, Erlangisafunctional programminglanguage:

```
-module(helloworld).
        -export([add/3,add/3,start/0]).

add(X, Y)->
  Z=X+Y,
  io:fwrite("~w~n",[Z]).

 add(X, Y, Z)-
   >A=X+Y+Z,
  io:fwrite("~w~n",[A]).

 start()->
   add(5, 8),
  add(5, 6, 8).
```

Outputoftheaboveprogramwilllooklike asshownbelow:

```
13
```

19

Passfunctionasparameterpython

Inthisarticle, wearediscussingthepass functionasaparameter inpython. Functionscantake multiple arguments. These arguments can be objects, variables (of the same or different data types), and functions. Python functions are the first elegant gadgets.

Within the following instance, a feature is assigned to a variable. This project no longer names a function. This takes the feature objectpointed to by "shout" and creates a second call, "yell," pointingto it.

Facts may be exceeded to features as arguments. Arguments are unique after the characteristic call-in parentheses. You could add as many arguments as possible; separate them with commas.

The following example has a function with one argument (fname). At the same time, the function is referred to as passing the name used in the characteristic to print the overall call.

Example 1: Right here, wegiveaninstanceofthe pass functionasaparameterinpython. Theexample is given below -

```
defmy_function(fname):
print(fname + " Id")
my_function("Emil")
my_function("Phone")
```

Result: We assemble the above program, and after compilation, we runthe program. Then the result is given below -

```
EmailId
PhoneId
```

Example 2: Right here, we give another instance of the pass function as a parameter in python. The example is given below -

```
defshout(text):
   return text.upper()
print(shout('HelloWorld'))
yell = shout
print(yell('Hello Coders'))
```

Result: We assemble the above program, and after compilation, werunthe program. Then the result is given below -

```
HELLOWORLD
```

```
HELLOCODERS
```

Wrapper Function: A wrapper function or decorator allows you to wrap another function and extendit without permanentlychanging the behavior ofthe wrapped function. Inthe decorator, the function is taken as an argument of another function and called inside the wrapper function.

Example 1: Right here, we give an instance of a wrapper function as a parameter in python. The example is given below -

```
defhello_decorator(func):
    definner1():
        print("Hellocoders,itisbeforethefunctionexecution") func
        ()
        print("Itisafterthefunctionexecution")
    returninner1
deffunction_to_be_used():
    print("Itisinsideofthisfunction")
function_to_be_used=hello_decorator(function_to_be_used)
function_to_be_used()
```

Result: We assemble the above program, and after compilation, werunthe program. Then the result is given below -

```
Hellocoders,itisbeforethefunctionexecutionIt is
inside of this function
Itisafterthefunctionexecution
```

Lambda wrapper function: InPython, a namelessfunctionapproachthatthe character hasno call. As you recognize, the def keyword defines ordinary functions, and the lambda keyword is used to create anonymous functions. This characteristic will have a varietyofarguments but evaluates and returns the simplest expression.

Alambdafunctionalsocanhaveeveryothercharacterasanargument.Thesubsequentexample suggests a primary Lambda characteristic surpassed every other Lambda function as a controversy.

Example 1: Right here, we give an instance of the lambda wrapper function as a parameter in python. The example is given below -

```
square = lambda a:a *
acube=lambdafunc:func**3
print("Thesquareof4is:"+str(square(4)))
print("\nThe cubeof"+str(square(4))+"is:"+str(cube(square (4))))
```

Result: We assemble the above program, and after compilation, werunthe program. Then the result is given below -

```
The square of 4
is:16Thecubeof16is:
4096
```

Higher order function: Since functions are objects, we can pass them as arguments to other features. Capabilities that take other features as arguments are also referred to as better-order functions. The subsequent instance creates a Greet feature that takes a feature as an issue.

Example 1: Right here, we give an instance of a higher order function as a parameter in python. The example is given below -

```
defshout(text):
    returntext.upper()
defwhisper(text):
    returntext.lower()
defgreet(function):
    greeting=function("Hello,weare createdbyahigherorderfunctionpassed asanargument.")
    print(greeting)
greet(shout)
greet(whisper)
```

Result: We assemble the above program and run the program after compilation. Then the result isgiven below -

```
HELLO,WEARE CREATEDBYAHIGHERORDERFUNCTIONPASSEDASANARGUMENT.
hello, wearecreated byahigherorderfunctionpassed asanargument.
```

Conclusion: In Python, you can pass function objects to other functions. Functions can be propagated inPython. Python has built-in functionsrequiring you to specifythe functionasone or more arguments so you can call it later.

map,filter,andreduceinPythonwithExamples

Python Streams

Python stream is a term for a particular paradigm for data processing that involves thesequentialprocessingofdataitemsastheypassthroughapipelineofprocesses. Streamsallowdata processing to be continuous, effective, and memory-friendly without loading the entire dataset into memory at once.

The map, filter, and reduce functions in Python are higher-order functions that work on sequences of data,suchaslistsorotheriterableobjects.Streamscanbeusedinconjunctionwiththesemethods.

With the help of these functions, you may quickly and effectively analyze typical data on sequence elements.

Whatisfunctionalprogramming?

Functional programming is a programming paradigm that treats computation as evaluating mathematical functions and avoids changing state and mutable data.

Infunctionalprogramming, functionsare first-classcitizens, meaningtheycanbeassignedto variables, passed as arguments to other functions, and returned as values from other functions.

Functionalprogramming emphasizes**immutability**, meaning that once a value is assigned, it cannot be changed, and it avoids sideeffects, whicharechangestothestateorbehaviourthat affect theresult ofa function beyond its return value.

Functionalprogrammingispossiblein Pythonusing anumberoffeaturesandtools,such as:

1. Higher-order functions- Python enables the assignment of functions to variables, the passing of functions as arguments, and the return of functions as values. Higher-order functions are the functions thatacceptotherfunctionsasargumentsorreturnthemasresultsandcanbeusedasaresultof this.Strong functionalprogramming techniques like giving functionsasparameters,returning functions from functions, and constructing functions on the fly are made possible by higher-order functions.

```
#Exampleofusing higher-orderfunctionsinPython #
Function that adds 1 to the passed value (x)
defadd(x):
   returnx+1

#Functionthatmultipliesthepassedvalue(x)by2
defmultiply(x):
   returnx*2

#Functionthatappliesanother functiononpassed value(x)
defapply(func,x):
   returnfunc(x)

result1 = apply(add, 3)# Result: 4
result2=apply(multiply,3)#Result:6
```

2. Lambda Functions- Lambda functions, also known as anonymous functions, can be defined inline without requiring a formal function declaration and are short and one-time-use functions. They are helpful for the performance of a single task that only requires one line of code to convey.

```
#ExampleofusinglambdafunctionsinPython
```

```
#Lambdafunctionthat adds1tox add
= lambda x: x + 1

#Lambdafunctionthatmultipliesxby2
multiply = lambda x: x * 2

result1 = add(3)# Result: 4
result2=multiply(3)#Result:6
```

3. map, filter, and reduce-Map, Filter, and Reduce are built-in Python functions that can be used for functional programming tasks. With the help of these operations, you may apply a specific function to sequence items using the 'map', filter sequence elements based on a condition using the 'filter', and cumulatively aggregate elements using the 'reduce'.

```
#Exampleofusing map,filter,andreduceinPython data =
[1, 2, 3, 4, 5]

#Usingthemaptoapplyafunctiontoeachelement #
Lambda function returns the square of x
result1=map(lambdax: x*2,data)#Result:[2,4,6,8,10]

#Usingthe filtertofilterelementsbasedonacondition #
Lambda function returns True for an even number
result2=filter(lambda x: x%2== 0,data)#Result: [2,4]

#Using reducetoaggregateelements
#Lambdafunctionreturnsproductofxandy
fromfunctoolsimport reduce
result3=reduce(lambda x,y:x*y,data)#Result:120
```

4. List Comprehension - List comprehensions are supported by Python, which are simple and expressive techniques to build new lists from older ones. Functional programming techniques such as mapping, filtering, and aggregating can be implemented using list comprehensions.

```
#Exampleofusinglist comprehensionsinPython data
= [1, 2, 3, 4, 5]

#Using list comprehensiontoapplyafunctiontoeachelement
result1 = [x * 2 for x in data]# Result: [2, 4, 6, 8, 10]
```

```
#Using list comprehensiontofilterelementsbasedonacondition
result2 = [x for x in data if x % 2 == 0]# Result: [2, 4]

# Using list comprehension to aggregate elements
result3=reduce(lambdax, y:x* y,data)#Result120
```

Let'sdivedeeperintothemap(),filter(),andreduce()functionsinPython.

1. map() -Python's map() methodappliesaspecified functionto eachitemofaniterable(suchas a list, tuple, or string) and then returns a new iterable containing the results.

The map()syntaxisasfollows:**map(function,iterable)**

The first argument passed tothe map function is itselfa function, and the second argument passed isan iterable (sequence of elements) such as a list, tuple, set, string, etc.

Example1-usageofthemap():

```
#Using map()to squareeachelementofthedatalist data
= [1, 2, 3, 4, 5]

#Mapfunctionreturnsthemapobject
squares = map(lambda x: x*x, data)

#Iteratingtheelementsofthesquares
foriinsquares:
   print(i,end="")

#Also,wecanconvertthemapobject intoalist squares =
list(map(lambda x: x*x, data)) print(f"Squares:
{squares}")
```

Output:

```
1, 4, 9, 16,25
Squares:[1,4,9,16,25]
```

Here, the map function takes each element one by one from the data starting from x = 1. Each element ispassedtothe lambda function, returning itssquare. Andthereturnedvalue isstoredinthe mapobject (an iterable).

2. filter() - The filter() function in Python filters elements from an iterable based on a given condition or function and returns a new iterable with the filtered elements.

Thesyntax forthefilter()isas follows:**filter(function,iterable)**

Here also, the first argument passed to the filter function is itself a function, and the second argumentpassed is an iterable (sequence of elements) such as a list, tuple, set, string, etc.

Example1-usageofthefilter():

Youaregivena list ofintegersandshould filtertheevennumbersfromthe list. #

```
Using filter() to filter even numbers from a list
data=[1,2,3,4,5]

#Thefilter functionfilterstheevennumbers fromthedata #
and returns a filter object (an iterable)
evens=filter(lambdax:x%2==0,data)

#Iteratingthevaluesofevens
foriinevens:
    print(i,end="")

#Wecanconvertthefilter objectintoalistasfollows:
evens=list(filter(lambdax:x%2==0,data))

#Printingtheevenslist
print(f"Evens={evens}")
```

Output:

```
24
Evens=[2,4]
```

Example2:FilteringPerfect Squares

Youaregivenalistofrandomintegersandshouldfiltertheperfectsquaresfromthem.Aperfect square is a number that can be expressed as the product of the same whole number.

```
#Pythonto demonstrateusageoffilterfunction
from mathimportsqrt

# List that contains random integer values
data=[0,1,4,6,8,9,10,12,16,81,23, 36]
```

```
#Functionthatreturnstruefor perfectsquares
defisPerfectSqr(i):
    returnsqrt(i).is_integer()

#Storing theresult
answer=list(filter(isPerfectSqr, data))

#Printing the result
print("Answer:",answer)
```

Output:

```
Answer:[0,1,4,9,16,81,36]
```

In the above example, we have a data list that contains some random integers. The 'isPerfectSqr' function returns True for perfect square numbers and False for others. The filter function filters out the numbers from the data, which is a perfect square, and returns an iterable that contains those perfect square numbers. In the end, we converted the result into a list and printed it into the console.

Example3:FilterOut NamesStarting withH

You are given a list containing names of persons and you should filter out the names starting with the letter 'H'. Below is the solution of the problem:

```
#Method1

#Using filter()to filternamesstartingwithletterH # A
list containing names
names=["Arun", "Sonu", "Harsh","Harry", "Anu", "Hassi"]

#Thefilter functionfiltersthename fromthenames #
and returns a filter object (an iterable)
# We can convert the filter object into a list as follows:
name_start_with_H=list(filter(lambdax:x[0]=='H',names))

#Printingthename_start_with_Hlist
print(f"Method1result={name_start_with_H}") #

Method 2

#Wecanalsouseafunction insteadoflambdafunction
#TheH_namefunctionreturnstrueifx(name)startswith 'H'
```

```
defH_name(x):
   returnx[0]=='H'

#Filteringtheresultandprintingit intotheconsole
name_start_with_H = list(filter(H_name, names))
print(f"Method 2 result = {name_start_with_H}")
```

Output:

```
Method1result=['Harsh','Harry','Hassi']Met
hod2result=['Harsh','Harry', 'Hassi']
```

In theaboveexample,thelambdaorH_namefunction returnstrueforeach xin namesstartingwith letter H. The filter function filters all the names which satisfies the condition.

3.reduce()- In Python, reduce() is a built-infunction thatapplies a given function to the elements of an iterable, reducing them to a single value.

Thesyntaxforreduce()isasfollows:**reduce(function,iterable[,initializer])**

The**functionargument** isafunctionthattakestwoargumentsandreturnsasinglevalue.Thefirst argument is the accumulated value, and the second argument is the current value fromthe iterable.

The**iterable**argumentisthesequenceofvaluestobe reduced.

Theoptionalinitializerargumentisusedtoprovideaninitialvaluefortheaccumulatedresult.Ifno initializer is specified, the first element ofthe iterable is used as the initial value.

Here'sanexamplethatdemonstrateshowto usereduce()tofind thesumofalist ofnumbers:

Example1:

Youaregivenalistcontainingsomeintegersandyoushouldfindthesum of all elementsinthelist using reduce function. Below is the solution of the problem:

```
#Examplestounderstandthereduce()function
fromfunctoolsimport reduce

#Functionthatreturnsthesumoftwonumbers
defadd(a,b):
   returna+b

#Our Iterable
num_list =[1, 2,3,4, 5, 6,7,8,9, 10]
```

```
#addfunctionispassedasthefirst argument,andnum_list ispassedasthesecondargument sum =
reduce(add, num_list)
print(f"Sumoftheintegersofnum_list:{sum}")

# Passing 10 as an initial value
sum=reduce(add,num_list,10)
print(f"Sumoftheintegersofnum_listwith initialvalue 10 :{sum}")
```

Output:

```
Sumoftheintegersofnum_list:55
Sumoftheintegersofnum_listwithinitial value10 :65
```

In the above example, the reduce function takes two elements 1 and 2 from the num_list and passes to theaddfunctioninthe first iteration. Theadd functionreturnsthesomeofthe1and2whichis3. Inthe second iteration, the reduce function passes the result of the previous call which is 3 and the next element which is also 3. This process is repeated untilall elements have been processed.

In the case, where we pass the initial value (10), the reduce function takes one element from the num_list and initial value (10) and passes to the add function in the first iteration.

Example2:Usingoperatorfunctionswithreduce function

In the below example, we have used operator.add to perform addition, operator.mul to perform multiplication and operator.concat to perform concatenation on strings.

```
#Pythonprogramtodemonstrate
#howto useoperatorfunctionswithreducefunction

#Importingreducefunction
fromfunctoolsimport reduce

#Importingoperator
importoperator

# Creating
listsmy_list1=[1,2,3,4,5
]
my_list2=["I","Love","Javatpoint"]

#Calculatingthesumofthenumbersofmy_list1 #
using reduce and operator.add
```

```
sum=reduce(operator.add,my_list1)

#Calculatingtheproductofthenumbersofmy_list1 #
using reduce and operator.mul
product=reduce(operator.mul,my_list1)

#Concatenatingalltheelementsinmy_list2 #
using reduce and operator.concat
concated_str1=reduce(operator.concat,my_list2)

#Wecanachievethesameoutputbyusingoperator.add
concated_str2 = reduce(operator.add, my_list2)

#Printing result
print(f"Sumofallelementsin my_list1:{sum}")
print(f"Product of all elements in my_list1 : {product}")
print(f"Concatenatedstringbyusingoperator.concat :{concated_str1}")
print(f"Concatenated string by using operator.add : {concated_str2}")
```

Output:

```
Sum of all elements in my_list1 :
15Productofallelementsinmy_list1:120
Concatenatedstringbyusingoperator.concat:ILoveJavatpointCon
catenated string by using operator.add : ILoveJavatpoint
```

Here,the functionargument inthe reduce function isreplaced withthe operator functions. Allthe steps are same as previous examples.

Example3: Find thelargestofthegivennumbers.

Inthisexample, youaregivena list ofintegersand youshould findthe largest numberusingthereduce function. Below is the solution of the problem:

Thearethreemethods,wecanusetoachievethesame result:

Method1-Usingnormalfunction

```
#Importing reducefunctionfromthefunctoolsmodule
fromfunctoolsimport reduce

#Alistcontainingsomeintegers
```

```
num=[20,22,24, 12,6, 88,10, 55,66]

"""Method 1-Usingsimplefunction"""

#Functionthatreturnsthelargestofxandy
deflarge(x,y):
    returnxifx>yelsey

#Usingreduceto findthelargest ofallandprintingtheresult largest
= reduce(large, num)
print(f"Largestfoundwithmethod1: {largest}")
```

Method2-UsingLambdafunction

```
"""Method2-Usinglambdafunction"""

#Usingreduceto findthelargest ofallandprintingtheresult largest
= reduce(lambda x, y: x if x > yelse y, num) print(f"Largest
found with method 2: {largest}")
```

Method3-Usingmaxfunction

```
"""Method3-Usingmax()function"""

#Usingreduceto findthelargest ofallandprintingtheresult largest
= reduce(max, num)
print(f"Largestfoundwithmethod3: {largest}")
```

Output:

```
Largestfoundwithmethod1:88Lar
gestfoundwithmethod2:88Larges
tfoundwithmethod3:88
```

In the above example, the large function, lambda function, and the max function returns the maximum ofxand ytothereducefunction. Andthereduce functionparsealltheelementsone byone. At theend, it returns the largest of all.

CONCLUSION:

Inconclusion, **map(), filter(), and reduce() are built-in functions in Python**that are commonlyused for functional programming.

map() is used to apply a given function to each element of an iterable and returns a new iterable with the results.

filter() is used to filter elements from an iterable based on a given condition or function and returns a new iterable with the filtered elements.

reduce() is used to apply a given function to the elements of an iterable in a cumulative way, reducing the iterable to a single value.

These functions or tools give you strong capabilities for processing data quickly and expressively, making it simple to convert, filter, and aggregate data. They are frequently used in Python's functional programming concepts to create readable and efficient codes.

PythonModules

In this tutorial, we will explain how to construct and import customPython modules. Additionally, we may import or integrate Python's built-in modules via various methods.

WhatisModularProgramming?

Modular programming is the practice of segmenting a single, complicated coding task into multiple, simpler, easier-to-manage sub-tasks. We call these subtasks modules. Therefore, we can build a bigger program by assembling different modules that act like building blocks.

Modularizingourcodeina bigapplicationhasa lotof benefits:

Simplification: A module often concentrates on one comparatively small area of the overall problem instead of the full task. We will have a more manageable design problem to think about if we are only concentrating on one module. Program development is now simpler and much less vulnerable to mistakes.

Flexibility: Modules are frequently used to establish conceptual separations between various problem areas. It is less likely that changes to one module would influence other portions of the program if modules are constructed in a fashion that reduces interconnectedness. (We might even be capable of editing a module despite being familiar with the program beyond it.) It increases the likelihood that a group of numerous developers will be able to collaborate on a big project.

Reusability: Functions created in a particular module may be readily accessed by different sections of the assignment (through a suitablyestablished api). As a result, duplicate code is no longer necessary.

Scope: Modules often declare a distinct namespace to prevent identifier clashes in various parts of a program.

In Python, modularization of the code is encouraged through the use of functions, modules, and packages.

What areModulesinPython?

A document with definitions of functions and various statements written in Python is called a Python module.

InPython,wecandefineamoduleinoneof3 ways:

Python itselfallowsforthecreationofmodules.

Similar to the re (regular expression) module, a module can be primarily written in C programming language and then dynamically inserted at run-time.

Abuilt-inmodule,suchastheitertoolsmodule,isinherently includedintheinterpreter.

A module is a file containing Python code, definitions of functions, statements, or classes. An example_module.py file is a module we will create and whose name is example_module.

We employ modules to divide complicated programs into smaller, more understandable pieces. Modules also allow for the reuse of code.

Rather than duplicating their definitions into several applications, we may define our most frequently used functions in a separate module and then import the complete module.

Let'sconstructamodule. Savethefileasexample_module.pyafterenteringthefollowing.

Example:

```
#Here, wearecreatingasimplePythonprogramtoshowhowto createamodule. #
defining a function in the module to reuse it
defsquare( number ):
  #here,theabove functionwillsquarethenumberpassedasthe input result
   = number ** 2
   returnresult      #here,wearereturning theresultofthefunction
```

Here, a module called example_module contains the definition of the function square(). The functionreturns the square of a given number.

HowtoImportModulesin Python?

InPython,wemayimportfunctionsfromonemoduleintoourprogram,oraswesayinto,another module.

For this, we make use ofthe import Pythonkeyword. Inthe Pythonwindow, we addthe next to import keyword, the name of the module we need to import. We will import the module we defined earlier example_module.

Syntax:

```
importexample_module
```

Thefunctionsthatwe definedin theexample_modulearenotimmediately imported into thepresent program. Only the name of the module, i.e., example_ module, is imported here.

Wemayusethedotoperatortousethefunctionsusing themodulename.Forinstance:

Example:

```
#here,wearecallingthemodulesquaremethodandpassingthevalue4 result =
example_module.square(4)
print("Byusingthemodulesquareofnumberis:",result)
```

Output:

```
Byusingthemodulesquareofnumberis:16
```

ThereareseveralstandardmodulesforPython.ThecompletelistofPythonstandardmodulesis available. The list can be seen using the help command.

Similartohowweimportedourmodule,auser-definedmodule,wecanuseanimportstatementto import other standard modules.

Importingamodulecanbedoneinavarietyofways.Belowisalistofthem.

Pythonimport Statement

UsingtheimportPythonkeywordandthedotoperator,wemayimportastandardmoduleandcan access the defined functions within it. Here's an illustration.

Code

```
#Here, wearecreatingasimplePythonprogramtoshowhowto importastandard module # Here,
we are import the math module which is a standard module
importmath
print("Thevalueofeuler's numberis",math.e)
#here,weareprinting theeuler'snumberfromthemathmodule
```

Output:

```
Thevalueofeuler'snumberis2.718281828459045
```

ClassesandObjectsinPython

You wanted to build a house. What is the first thing you do to start the building process? You create a plan on how you want your house. You follow the plan to build the house. A plan is like a blueprint of the house that is not built yet and will be built based on it.

Why do we even need a plan? For the organization of all components like different rooms, walls, windows, and doors in the right places with the correct dimensions.

Definitionofaclass:

In any programming language**, a class is a user-defined plan or blueprint using which objects or instances of the class are created.**

You may wonder why we need classes in programming. We can create something like a variable or a structure, store what we want, and use it.

Aclasscanhaveitsattributes-variablesand functions withpre-stored valuesandfunctionalities.

Example situation: If we want to store the data of different age groups of children and their details in an orphanage :

We cannot just create a list and keep on storing the ages and names of the children without any organization.

Creating multiple listsfor multiple age groups -one list to store ages,one fornames, anotherto match -becomes complex and leads to ambiguity.

Syntax:

#Definitionofaclass

```
class class_name:
   #bodyoftheclass
   #variables
   #functions
      #bodyofthefunctions
```

Importantpoints:

"**class**"isakeywordinpythonlibraries. A

class name must be **capitalized**.

Attributesare the variables owned by the class (declared inside the class) that the created objects can use.

Methodsarethefunctionsownedbytheclass(defined insidetheclass) thatthecreatedobjectscanuse. The

created objects using the dot (.) operator uses these attributes and methods of a class.

Exampleprogram:

```
class Employee:
   #Attributeslikename,id
   #Methods
```

Now,let'sgettheoverviewof Objects:

OBJECTS-The instancesofaclass

Now, we have the blueprint. This is the time for action and implementing the idea of the class. The house - the plan is ready. It is time to start the construction. The organizing pattern is ready. Now, we build the rooms one by one. You can assume one of the objects of the house class is a room. We can create many objects-many different rooms. An object is like a specimen of the class.

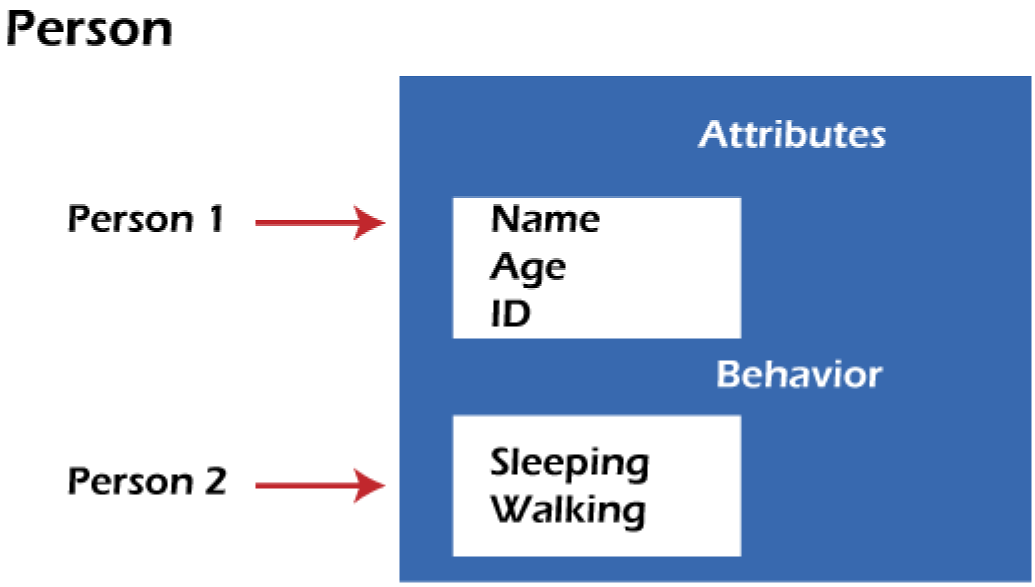
Thecharacteristicsofan object:

State: The stateofanobject refers to the attributes ofthe object - different variables withdifferent info about the object.

Identity: Itistheobject'snameto identifyeveryobjectuniquelyfromanother.

Behavior: Thebehaviorofanobjectreferstothemethodsofobject-differentfunctionalities.

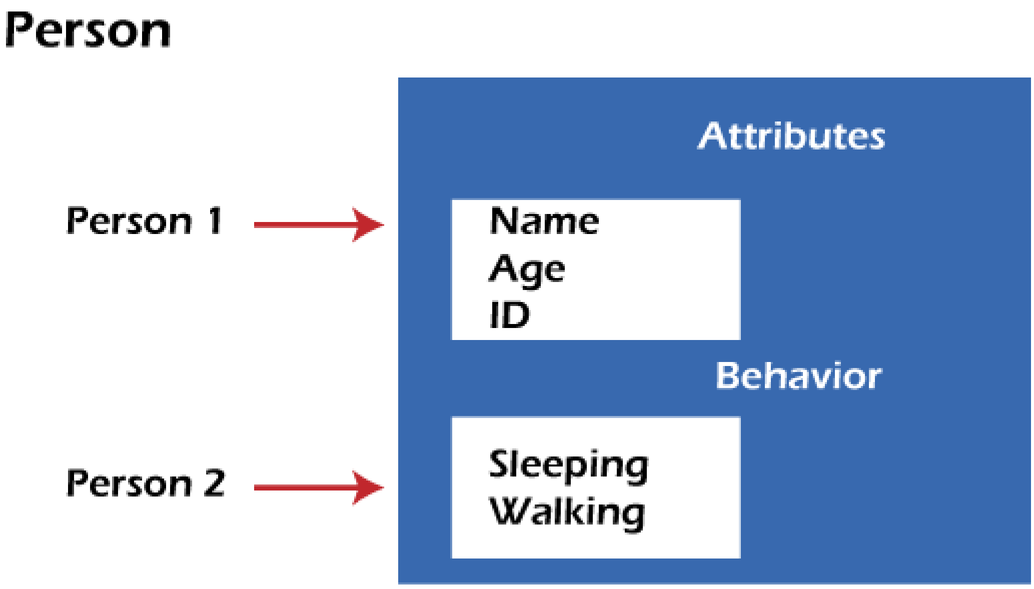
Examplesforthecharacteristicsofan object(Random):



Object	Identity	State	Behavior
Aperson	Rakesh	Literate,qualified, vegetarian	Speaking,walking,reading
Ashopping sale	Republicdaysale	Started,offers,Ended,Cancelled	Earnloyaltypoints

If we created a class, say, people, a person is an object in that class. The person's name is the unique identityto identify the object; variables like literate and vegetarian refer to the object's state - different attributes of the object. The functions like walking; sleeping explains the person's behavior (object).

Iftheobjectisadog:



Declaringan object:

Object_name=Class_name()

Depending onthe need, wecancreateas manyobjects as wewant for asingle function. So, wecreate a class and store the blueprint to do a task or organize something; we store them in variables called attributes and functions called methods.

Then, intheprogram, whenwe needthe functionalityofaclassor itsattributesand methods, weaccess them by declaring an object of that class.

Syntaxtoaccess:

```
#Declaring
Object_name=Class_name()
```

```
#To access a class-attribute
Object_name.class_attribute=value

#To access a class-method
Object_name.method_name(arguments)
```

Letustakethe example above-ADog.

Program:

```
class Our_Puppy:
    Name="Snoopy"
    Color = "Brown"
    Breed="GermanSheppard"
    Hungry = "yes"

Snoopy=Our_Puppy()
print("Thecolorofmydogis:",Snoopy.Color)
```

Output:

```
Thecolorofmydog is: Brown
```

Now,goingtothemethodsofaclass,weneedtolearnaboutthe **selfvariable** and **theinit method.**

The**init()**methodis a constructorwhich willbeinvoked by defaultwhen anew objectis created and initialized.

"self"isnotakeywordinpython.Itisaspecialvariable.Whenwedeclareamethodinaclass,we need to declare the first argument as self.

We will be able to access the attributes and the methods of the class using this self-argument. It holdsthe arguments to the attributes.

Eveninthe initmethod,wemust declarethefirstargument asself.

Self determinesthecurrentinstanceof theclass;In thecaseof theinitmethod,itreferstothenewly created object, while in other methods, it refers to the object whose method is called.

Evenifwewanttocreatea methodwithnoarguments,weneedtokeep theselfvariableastheonly argument in the method.

Thesyntaxwilllook likethis:

```
class Class_Name:
   attribute(variable)1
   attribute(variable)2
   ...
   attribute(variable)N

   definit(self,arg1,...,argn):

   defMethod_name1(self,arg1,...,argn):

   defMethod_name2(self):
   ...
   defMethod_nameN(self,arg1,...,argn):
```

Exampleprograms:

```
#Usinginitmethod

classStudent:
   definit(self,name,age,email):
      self.name = name
      self.age    =    age
      self.email=email
name = input("Please enterthe name ofthe student1: ")
age=int(input("Pleaseentertheageofthestudent1:"))   stud
= Student(name,age,'santhosh@gmail.com')
name = input("Please enterthe name ofthe student2: ")
age=int(input("Pleaseentertheageofthestudent2:"))
stud2 = Student(name,age,")
print("Stud_1.name=",stud.name)
print("Stud_2.name=",stud2.name)
```

Output:

```
Pleaseenterthenameofthestudent1:SanthoshPlease
enter the age of the student1: 19
Pleaseenterthenameofthestudent2:RakeshPlease
enter the age of the student2: 19
```

```
Stud_1.name=SanthoshStu
d_2.name = Rakesh
```

```
#Usingmoremethods intheclass
classPerson_details:
    definit(self,name):
        self.name = name
    def setName(self,name):
        self.name = name
    defgetName(self):
        returnself.name

name= input("Enterthenameoftheperson1:") p1 =
Person_details(name)
name=input("Enterthenameoftheperson2:") p2 =
Person_details(name)
print("PersonP1name:",p1.getName())
print("PersonP2name:",p2.getName())
```

Output:

```
Enterthenameoftheperson1:MaheshEnt
er      the      name      ofthe      person2:
RakeshPerson P1 name: Mahesh
PersonP2name: Rakesh
```

Attributesforan Empty class:

Python language allows the programmerto create attributes of an objectdeclaredforan empty class. We can even modify the values of attributes in different objects.

Here is an example of an empty class "Student_info". In the body ofthe program, we created an object Stud_1 for the class and gave values to 3 attributes for the class:

```
classStudent_info:
    pass

Stud_1 =Student()
Stud_1.name='Sonu'
Stud_1.age = 19
```

```
Stud_1.graduate = 'B-
tech'print("Stud_1.name:",Stud_1.name)
print("Stud_1.age:", Stud_1.age)
print("Stud_1.graduate:",Stud_1.graduate)
```

Output:

```
Stud_1.name:
SonuStud_1.age:
19Stud_1.graduate:B-tech
```

Wediscussedabovethattheinitmethodisaconstructor.Divingdeepinto**Constructorsinpython:**

Aconstructorcanbeunderstoodasatypeofamethod.Weusethistoperformactionsortaskslike initializing variables and other tasks that **must bedone when a new object is created.**

Inpython,weusethesameinit(self)inallclasses.The conventions of the name:Twoleading and trailing underscores. It is designed this wayto prevent ambiguity with user-defined methods.

Ifwecreateaclasswithout using theconstructor,bydefault, pythoncreatesaconstructor whichdoesn't have functionality other than instantiating the created object.

Let'ssummarizethelearnedtopictillnow:

Everyclassinpythonwillconsistof3keypartners:

The constructor

Attributesoftheclass

Methods of the class

Wecreateaclass.Insidetheclass,webuildtheconstructorinit()withselfargumentandother argumentsthatwebindwiththeclassattributesinsidethebody.So,whenwecreateanobjectwith some parameters, these parameters will occupythe arguments and be stored in the class attributes.

It isthe same innormalmethods, but innormalmethods,wewillhave ataskgoing on, but inthe init,it is only about initializing the arguments.

Theselfkeywordhelpsbindtheparametersandargumentsoftheconstructorandothermethods.It must be declared as the first argument in any method.

PythonInstance

Introduction

Python is a type ofprogramming language used bydevelopers worldwide. One ofthe essential features ofPythonisobject-orientedprogramming(OOP).Itallowtheprogrammertocreateobjects,classes, and instances. In this article, we are going to discuss Python instances in detail and demonstrate how they work with an example.

UnderstandingObject-Oriented Programming

Before we know about Python instances, it's very much important to understand the basics of object-oriented programming. OOP is a programming concept that revolves around objects, which are instances ofclasses. Classes are user-defined datatypes that encapsulate data and functions that operate on that data.

In OOP, objects are created from classes, and each object is unique. You can create multiple objects from the same class, and each object will have its own data and behavior. The following code shows how to define a class in Python:

Program1:

```
classMyClass:
   definit(self, name,age): self.name =
      name
      self.age=age

   defget_info(self):
      returnf"{self.name}is{self.age}yearsold."

#createaninstanceofMyClass
my_obj=MyClass("John",25)

#callget_infomethod toprint outtheinformation
print(my_obj.get_info())#output:Johnis25yearsold.
```

Output:

```
John is 25 years old.
```

Explanation

The above code defines a class MyClass with an initmethod . The init method initializes two instance variables name and age. The get_info method returns a formatted string that includes thevalues of the name and age instance variables.

Aninstanceoftheclassiscreatedwiththename"John"andage25.Then,theget_info method iscalled onthe instance, which prints the information "John is 25 years old." to the console.

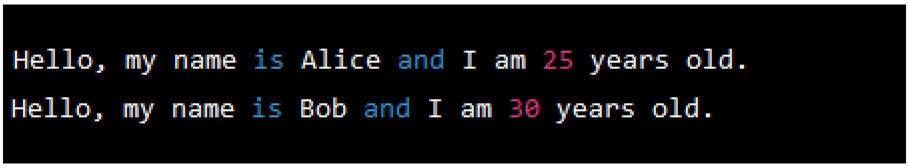
CreatingPython Instances

Once the programmer have defined a class, then they can create objects (or instances) from that class. To create an instance, the programmer simply call the class and assign the result to a variable. The following code snippet shows how to create an instance of the MyClass class:

Program2:

```
classPerson:
  definit(self, name,age): self.name =
    name
    self.age=age

  defgreet(self):
    print(f"Hello,mynameis{self.name}andIam{self.age}yearsold.")

#createinstancesofPersonclass
person1 = Person("Alice", 25)
person2 = Person("Bob", 30)

# callthe greetmethodoneachinstance
person1.greet()#output:Hello, myname isAlice andIam25 yearsold.
person2.greet() # output: Hello, my name is Bob and I am30 years old.
```

Output:



```
Hello, my name is Alice and I am 25 years old.
Hello, my name is Bob and I am 30 years old.
```

Explanation

The above code defines a Person class with a constructor that initializes name and age instance variables. The class also has a greet method that prints out a greeting with the name and age of the person.

Two instances of the Person class are created with different name and age values. Then, the greet method iscalledoneachinstance, whichprintsoutthegreetingwiththecorresponding values for name and age.

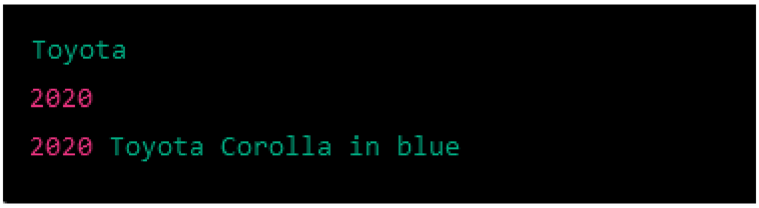
AccessingInstanceAttributesandMethods

Once the programmer have created an instance, then the programmer can access its attributes and methodsusingthedot notation. The followingcodesnippet showshowto accessthe nameattributeand call the get_info method of the person object:

Program3:

```
classCar:
    definit(self, make, model, year,color): self.make =
        make
        self.model=model
        self.year = year
        self.color = color

    defget_info(self):
        returnf"{self.year}{self.make}{self.model}in{self.color}"

#createaninstance ofCar
my_car=Car("Toyota","Corolla",2020, "blue")

# access instance attributes
print(my_car.make)#output:Toyota
print(my_car.year)# output: 2020

# call instance method
car_info=my_car.get_info()
print(car_info)#output: 2020ToyotaCorollainblue
```

Output:



```
Toyota
2020
2020 Toyota Corolla in blue
```

Explanation

TheabovecodeshowsaclassCar withaninitmethodthat initializes four instance variables make, model, year, and color. The get_info method returns a formatted string that includes the values of the year, make, model, and color instance variables.

An instance of the class is created with the make "Toyota", model "Corolla", year 2020, and color "blue". Then, the make and year instance variables are accessed using dot notationto print their values.

Finally, the get_info method is called on the instance and the returned string is stored in a variable car_info which is then printed to the console.

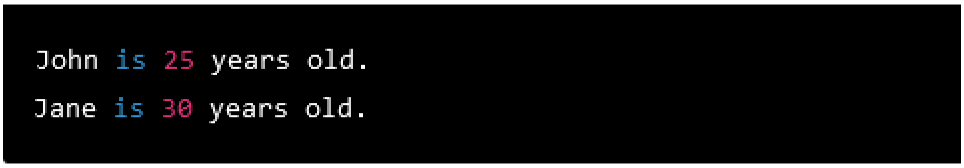
ModifyingInstanceAttributes

The programmer can also modify the attributes of an instance after it has been created. The following code snippet shows how to modify the attribute of the object:

Program4:

```
classMyClass:
  definit(self, name,age): self.name =
    name
    self.age=age

  defget_info(self):
    returnf"{self.name}is{self.age}yearsold."

#createaninstanceofMyClass
my_obj=MyClass("John",25)

#printtheinitialinformation
print(my_obj.get_info())#output:Johnis25yearsold.

#modifytheinstanceattributes
my_obj.name = "Jane"
my_obj.age = 30

#printthe modifiedinformation
print(my_obj.get_info())#output:Janeis30years old.
```

Output:



```
John is 25 years old.
Jane is 30 years old.
```

Explanation

The above code shows a MyClass class with an initmethod that initializes two instance variables name and age. Here the get_info method returns a formatted string that includes the values ofthe name and age instance variables.

An instance of the class is created with the name "John" and age 25. The initial information is printed by calling the get_info method on the instance.

The instance attributes name and age are then modified by directly accessing them using the instance name and the dot notation.

Finally, the modified information isprinted bycalling the get_info method onthe instance again, and it prints the modified information "Jane is 30 years old." to the console.

Benefitofusing PythonInstance

There are so many advantages of using Python instances in the programming projects. These are as follows.

Encapsulation: Python instances allow the programmer to encapsulate data and functions that operate onthat data. This means that the programmer can grouprelated data and functionalitytogether, making the code more organized and modular.

Reusability: Once the programmer have defined a class, then the programmer can create multiple instances ofthat class. This allows the programmer to reuse code and avoid duplicating functionality in the program.

Customization: Each instance of a class is unique and can be customized to suit specific needs. This means that the programmer can createobjectsthat have different attributes and behavior based ontheir specific use case.

Inheritance: In Python, the programmer can create subclasses that inherit attributes and behavior from a parent class. This allows the programmer to create more specialized classes that build on the functionality of existing classes.

Polymorphism: Polymorphism is a programming concept that allows the programmer to use the same codetooperateondifferenttypesofobjects. Pythoninstancessupport polymorphism, whichmeansthat the programmer can write code that works with different types of objects as long as they share a common interface.

Disadvantagesofusing PythonInstance

Therearealsosomanydisadvantagesofusingpythoninstances.Theseareasfollows.

Overhead: when the programmer wants to create instances for a class, it can be computationally expensive, especiallyifthe class has complexdata structuresor methods. This overhead can impact the performance of the code, particularly in applications that require high-speed processing.

Memory Usage: Each instance of a class requires memory to store its data and methods. If the programmer creates many instances of a class, it can consume a significant amount of memory, which may cause the application to slow down or crash.

Complexity: Object-oriented programming can be more complex than other programming paradigms, especiallyfor beginners. Understanding howto create classesand instances, and how theyinteract with each other, requires a more comprehensive understanding of programming concepts.

Maintenance: As the code grows, managing instances and their relationships can become more challenging. Code changes to a class can impact the behavior of all instances of that class, which can make it harder to maintain and debug code.

Design: Touseinstanceseffectively, theprogrammer needto design your classesandobjectscarefully. Poor design decisions can lead to a codebase that is difficult to understand and maintain.

Conclusion

Python instances are objects created fromclasses that encapsulate data and functions. The programmer can create multiple instances from the same class, and each instance will have its own data and behavior. Python instances are a powerful feature of object-oriented programming that allows the programmer to modelreal-world objectsand theirbehavior incode.Bymastering Pythoninstances,the programmer will be able to write more modular, maintainable, and scalable Python code.

UNIT-V

DatabaseProgramming–Introduction-BasicDatabaseOperationsandSQL-Exampleofusing Database Adapters, Mysql - Regular Expression – Special Symbols and Characters – REs and Python

Whatis Database

The database is a collectionof inter-related data which is used to retrieve, insert and delete the data efficiently. Itisalso usedtoorganizethedataintheformofatable, schema, views, andreports,etc.

Forexample: ThecollegeDatabaseorganizesthedataabouttheadmin,staff,studentsand facultyetc. Using the

database, you can easily retrieve, insert, and delete the information.

DatabaseManagementSystem

Databasemanagementsystemisasoftwarewhichisusedtomanagethedatabase.For example: MySQL,Oracle, etc are a very popular commercial database which is used in different applications.

DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.

It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

DBMSallowsusersthefollowingtasks:

Data Definition: It is used for creation, modification, and removal of definition that defines the organization of data in the database.

DataUpdation: Itis usedfortheinsertion,modification,anddeletionoftheactualdatainthedatabase.

Data Retrieval: It is used to retrieve the data from the database which can be used by applications for various purposes.

UserAdministration: It isusedfor registeringandmonitoringusers, maintaindataintegrity, enforcing data security, dealing with concurrency control, monitoring performance and recovering information corrupted by unexpected failure.

Characteristicsof DBMS

Itusesadigitalrepositoryestablishedonaservertostoreand managethe information. It can

provide a clear and logical view of the process that manipulates data.

DBMScontainsautomaticbackupandrecoveryprocedures.

ItcontainsACIDpropertieswhichmaintaindatainahealthystateincaseoffailure. It can

reduce the complex relationship between data.

Itisusedtosupport manipulationandprocessingofdata. It is

used to provide security of data.

Itcanviewthedatabasefromdifferentviewpointsaccording totherequirementsoftheuser.

Advantages of DBMS

Controls database redundancy: It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.

Data sharing: In DBMS, the authorized users of an organization can share the data among multiple users.

EasilyMaintenance: Itcanbeeasilymaintainableduetothecentralizednatureofthedatabase system.

Reducetime: Itreducesdevelopmenttimeand maintenanceneed.

Backup: Itprovidesbackupandrecoverysubsystemswhichcreateautomaticbackupofdata from hardware and software failures and restores the data if required.

multiple user interface: It provides different types of user interfaces like graphical user interfaces, application program interfaces

DisadvantagesofDBMS

Cost of Hardware and Software: It requires a high speed ofdata processor and large memorysize to run DBMS software.

Size: Itoccupiesalarge space ofdisksandlargememorytorunthemefficiently.

Complexity: Databasesystemcreatesadditionalcomplexityandrequirements.

Higher impact of failure: Failure is highly impactedthe database because in most ofthe organization, allthe data stored ina single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

Database Connection

Inthissectionofthetutorial, wewilldiscussthestepsto connectthepythonapplicationtothedatabase. There are

the following steps to connect a python application to our database.

Importmysql.connectormodule

Createtheconnectionobject.

Create the cursor object

Execute the query

Creatingtheconnection

To create a connection between the MySQL database and the python application, the connect() method of mysql.connector module is used.

Pass the database details like HostName, username, and the database password in the method call. The method returns the connection object.

Thesyntaxtousetheconnect() isgiven below.

```
Connection-Object=mysql.connector.connect(host=<host-
name> , user = <username> , passwd = <password> )
```

Considerthefollowingexample.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn=mysql.connector.connect(host="localhost", user ="root",passwd="google")

#printingtheconnectionobject
print(myconn)
```

Output:

```
<mysql.connector.connection.MySQLConnectionobjectat0x7fb142edd780>
```

Here,wemustnoticethatwecanspecifythedatabasenameintheconnect()methodifwewantto connect to a specific database.

Example

```
importmysql.connector
```

```
#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="m ydb")

#printingtheconnectionobject
print(myconn)
```

Output:

```
<mysql.connector.connection.MySQLConnectionobjectat0x7ff64aa3d7b8>
```

Creatingacursorobject

The cursor object can be defined as an abstraction specified in the Python DB-API 2.0. It facilitates us to have multiple separate working environments through the same connection to the database. We can createthe cursor object by calling the 'cursor' function ofthe connection object. The cursor object is an important aspect of executing queries to the databases.

Thesyntaxtocreatethecursorobjectisgivenbelow.

```
<my_cur>=conn.cursor()
```

Example

```
import mysql.connector
#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="m ydb")

#printingtheconnectionobject
print(myconn)

#creatingthecursorobject
cur = myconn.cursor()

print(cur)
```

Output:

```
<mysql.connector.connection.MySQLConnectionobjectat0x7faa17a15748>
MySQLCursor:(Nothingexecutedyet)
```

Creatingnew databases

Inthissectionofthe tutorial,wewillcreate the new database PythonDB.

Gettingthelistofexisting databases

Wecanget thelistofallthedatabasesbyusing thefollowing MySQL query.

```
>showdatabases;
```

Example

```
importmysql.connector

#Createtheconnectionobject
myconn=mysql.connector.connect(host="localhost", user ="root",passwd="google")

#creatingthecursorobject
cur = myconn.cursor()

try:
    dbs=cur.execute("showdatabases")
except:
    myconn.rollback()
forxincur:
    print(x)
myconn.close()
```

Output:

```
('EmployeeDB',)
('Test',)
('TestDB',)
('information_schema',)('jav
atpoint',)
('javatpoint1',)
('mydb',)
('mysql',)('performance_sch
ema',)('testDB',)
```

Creatingthenew database

Thenewdatabasecanbecreated byusing thefollowingSQL query.

```
>createdatabase<database-name>
```

Example

```
import mysql.connector

#Createtheconnectionobject
myconn=mysql.connector.connect(host="localhost", user ="root",passwd="google")

#creatingthecursorobject
cur = myconn.cursor()

try:
   #creating a new database
   cur.execute("createdatabasePythonDB2")

   #gettingthe list ofallthedatabaseswhichwillnowincludethenewdatabasePythonDB dbs =
   cur.execute("show databases")

except:
   myconn.rollback()

forxincur:
      print(x)

myconn.close()
```

Output:

```
('EmployeeDB',)
('PythonDB',)
('Test',)
('TestDB',)
('anshika',)('information_sch
ema',)('javatpoint',)
('javatpoint1',)
```

```
('mydb',)
('mydb1',)
('mysql',)('performance_sch
ema',)('testDB',)
```

Creatingthetable

In this section of the tutorial, we will create the new table Employee. We have to mention the database name while establishing the connection object.

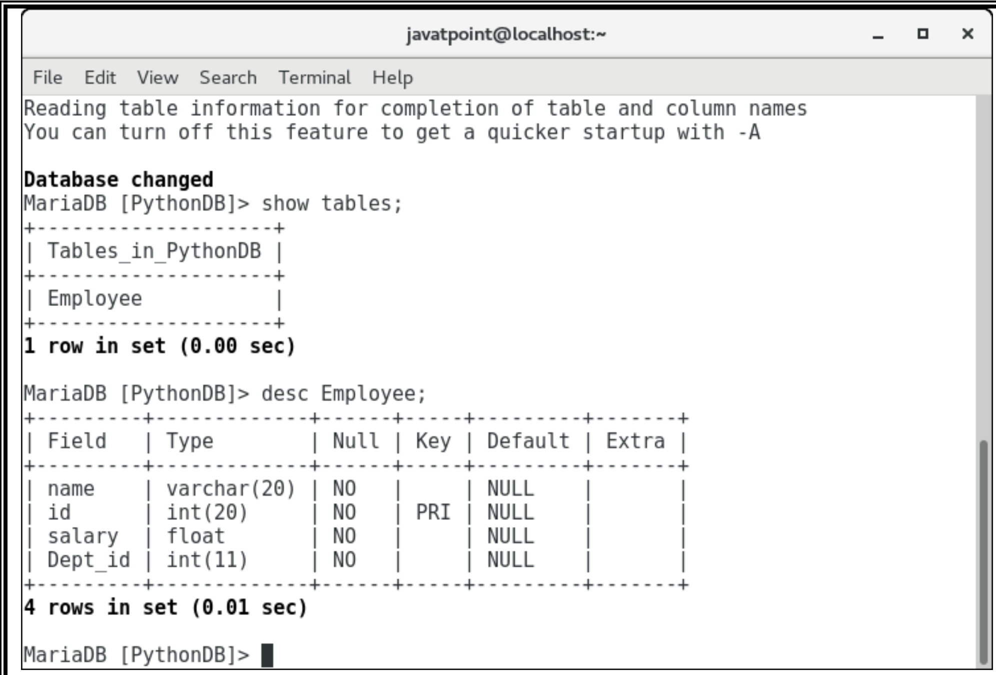
We can create the new table by using the CREATE TABLE statement of SQL. In our database PythonDB, the table Employee will have the four columns, i.e., name, id, salary, and department_id initially.

The followingqueryisusedtocreatethenewtableEmployee.

```
>createtableEmployee(namevarchar(20) notnull, id int primarykey, salaryfloat notnull, Dept_Idi nt not null)
```

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="PythonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
   #CreatingatablewithnameEmployeehaving fourcolumns i.e., name, id, salary, anddepartment id
   dbs=cur.execute("createtableEmployee(namevarchar(20)notnull, id int(20)notnullprimarykey,
salaryfloatnotnull,Dept_idintnot null)")
except:
   myconn.rollback()

myconn.close()
```



```
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [PythonDB]> show tables;
+--------------------+
| Tables_in_PythonDB |
+--------------------+
| Employee           |
+--------------------+
1 row in set (0.00 sec)

MariaDB [PythonDB]> desc Employee;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | NO   |     | NULL    |       |
| id      | int(20)     | NO   | PRI | NULL    |       |
| salary  | float       | NO   |     | NULL    |       |
| Dept_id | int(11)     | NO   |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
4 rows in set (0.01 sec)

MariaDB [PythonDB]> 
```

Now,wemaycheckthatthetableEmployeeispresentinthe database.

AlterTable

Sometimes, we may forget to create some columns, or we may need to update the table schema. The alter statement usedto alterthetable schema ifrequired. Here, wewilladd the column branch_name to the table Employee. The following SQL query is used for this purpose.

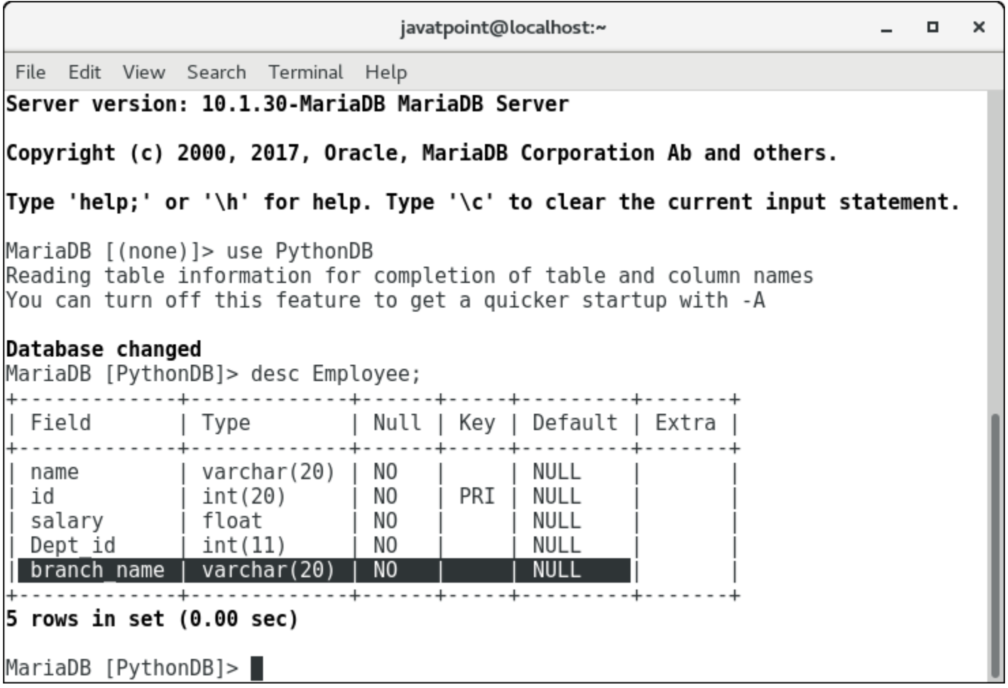
altertableEmployeeaddbranch_namevarchar(20)**not**null

Consider the following example.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
   #addingacolumnbranchnametothetable Employee
```

```
    cur.execute("altertableEmployeeaddbranch_namevarchar(20)notnull")
except:
    myconn.rollback()

myconn.close()
```



```
Server version: 10.1.30-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> use PythonDB
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [PythonDB]> desc Employee;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| name        | varchar(20) | NO   |     | NULL    |       |
| id          | int(20)     | NO   | PRI | NULL    |       |
| salary      | float       | NO   |     | NULL    |       |
| Dept_id     | int(11)     | NO   |     | NULL    |       |
| branch_name | varchar(20) | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

MariaDB [PythonDB]> 
```

InsertOperation

Adding arecordtothetable

The**INSERTINTO** statementisusedtoaddarecordtothetable.Inpython,wecanmentionthe format specifier (%s) in place of values.

Weprovidetheactualvalues inthe formoftuple intheexecute()methodofthecursor.

Consider the following example.

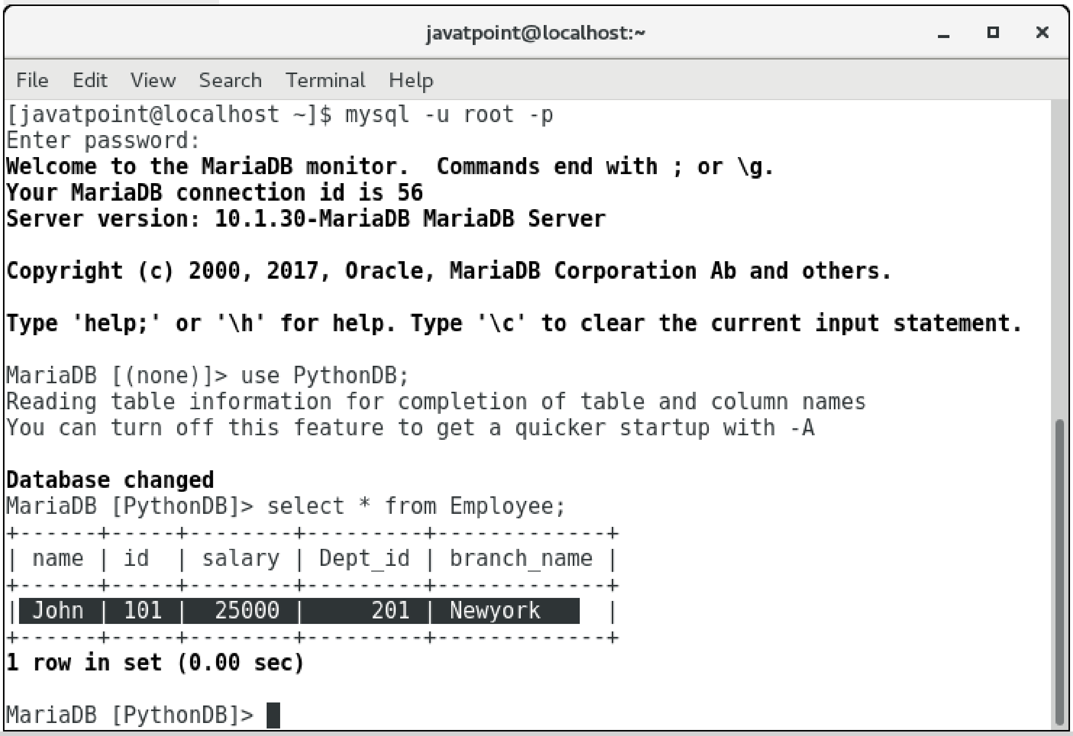
Example

```
import mysql.connector
#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")
#creatingthecursorobject
cur = myconn.cursor()
sql="insertintoEmployee(name,id, salary, dept_id, branch_name)values(%s, %s,%s,%s, %s)"
```

```
#Therowvaluesareprovided inthe formoftuple val =
("John", 110, 25000.00, 201, "Newyork")

try:
   #insertingthevaluesintothetable
   cur.execute(sql,val)

   #committhetransaction
   myconn.commit()

except:
   myconn.rollback()

print(cur.rowcount,"recordinserted!")
myconn.close()
```

Output:

1 recordinserted!



```
[javatpoint@localhost ~]$ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 56
Server version: 10.1.30-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> use PythonDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [PythonDB]> select * from Employee;
+------+-----+--------+---------+-------------+
| name | id  | salary | Dept_id | branch_name |
+------+-----+--------+---------+-------------+
| John | 101 |  25000 |     201 | Newyork     |
+------+-----+--------+---------+-------------+
1 row in set (0.00 sec)

MariaDB [PythonDB]> 
```

Insertmultiplerows

We can alsoinsert multiple rows at once using the python script. The multiple rows are mentioned as the list of various tuples.

Each element of the list is treated as one particular row, whereas each element of the tuple is treated as one particular column value (attribute).

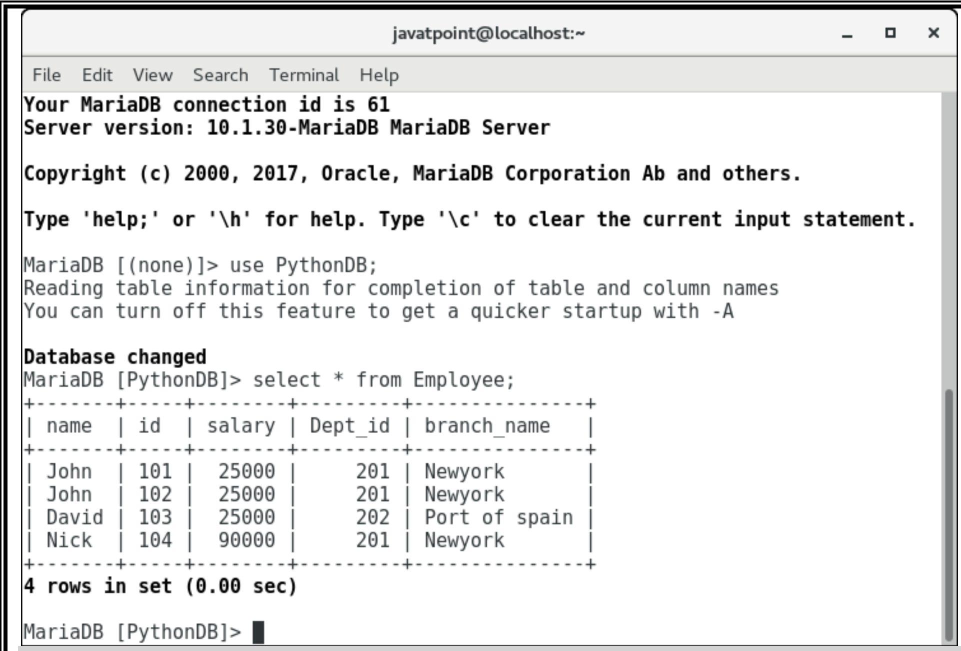
Considerthefollowingexample.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="PythonDB")

#creatingthecursorobject
cur = myconn.cursor()
sql="insertintoEmployee(name,id, salary, dept_id, branch_name)values(%s, %s,%s,%s, %s)"
val=[("John", 102, 25000.00,201,"Newyork"),("David",103,25000.00,202,"Portofspain"),("Nick",104,90000.00,201,"Newyork")]

try:
   #insertingthevaluesintothetable
   cur.executemany(sql,val)

   #commit the transaction
   myconn.commit()
   print(cur.rowcount,"records inserted!")

except:
   myconn.rollback()

myconn.close()
```

Output:

```
3records inserted!
```



```
Your MariaDB connection id is 61
Server version: 10.1.30-MariaDB MariaDB Server

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> use PythonDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [PythonDB]> select * from Employee;
+-------+-----+--------+---------+---------------+
| name  | id  | salary | Dept_id | branch_name   |
+-------+-----+--------+---------+---------------+
| John  | 101 |  25000 |     201 | Newyork       |
| John  | 102 |  25000 |     201 | Newyork       |
| David | 103 |  25000 |     202 | Port of spain |
| Nick  | 104 |  90000 |     201 | Newyork       |
+-------+-----+--------+---------+---------------+
4 rows in set (0.00 sec)

MariaDB [PythonDB]> 
```

RowID

InSQL, aparticular row is represented byan insertion id which is knownas rowid. We cangetthe last inserted row id by using the attribute lastrowid of the cursor object.

Considerthefollowingexample.

Example

```
import mysql.connector
#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")
#creatingthecursorobject
cur = myconn.cursor()

sql="insert intoEmployee(name, id,salary,dept_id,branch_name)values(%s,%s,%s,%s,%s)" val =

("Mike",105,28000,202,"Guyana")

try:
   #insertingthevaluesintothetable
   cur.execute(sql,val)

   #committhetransaction
```

```
    myconn.commit()

    #gettingrowid
    print(cur.rowcount,"recordinserted!id:",cur.lastrowid)

except:
    myconn.rollback()

myconn.close()
```

Output:

```
1record inserted!Id: 0
```

Read Operation

The SELECTstatementis usedtoread the valuesfrom the databases.We can restrictthe outputof a select query by using various clause in SQL like where, limit, etc.

Python provides the fetchall() method returns the data stored inside the tablein theform of rows. We can iterate the result to get the individual rows.

In this section of the tutorial, we will extract the data from the database by using the python script. We will also format the output to print it on the console.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
    #Reading the Employee data
    cur.execute("select*fromEmployee")

    #fetchingtherowsfromthecursorobject
    result = cur.fetchall()
    #printingtheresult
    forxinresult:
        print(x);
except:
    myconn.rollback()

myconn.close()
```

Output:

```
Backward Skip 10sPlay VideoForward Skip 10s
[redacted]('John', 101, 25000.0, 201, 'Newyork')
('John', 102, 25000.0, 201, 'Newyork')
('David',103,25000.0,202,'Portofspain')
('Nick',104,90000.0,201,'Newyork')
('Mike',105, 28000.0, 202, 'Guyana')
```

Readingspecificcolumns

Wecanreadthespecificcolumnsbymentioningtheir names insteadofusingstar (*).

Inthe following example, we willread the name, id, and salaryfromthe Employee table and print it on the console.

Example

```
import mysql.connector
#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")
#creatingthecursorobject
cur = myconn.cursor()try:
    #ReadingtheEmployeedata
    cur.execute("selectname,id,salaryfromEmployee")

    #fetchingtherowsfromthecursorobject
    result = cur.fetchall()
    #printingtheresult
    forxinresult:
        print(x);
except:
    myconn.rollback()
myconn.close()
```

Output:

```
('John', 101, 25000.0)
('John', 102, 25000.0)
('David', 103,25000.0)
('Nick',104,90000.0)
('Mike',105, 28000.0)
```

Thefetchone() method

The fetchone() method is used to fetch only one row fromthe table. The fetchone() method returns the next row of the result-set.

Considerthefollowingexample.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
    #ReadingtheEmployeedata
    cur.execute("selectname,id,salaryfromEmployee")

    #fetchingthefirstrowfromthecursorobject result =
    cur.fetchone()

    #printingtheresult
    print(result)
```

```
except:
    myconn.rollback()

myconn.close()
```

Output:

```
('John', 101, 25000.0)
```

Formattingtheresult

We can format the result by iterating over the result produced bythe fetchall() or fetchone() method of cursor object since the result exists as the tuple object which is not readable.

Considerthefollowingexample.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:

   #ReadingtheEmployeedata
   cur.execute("selectname,id,salaryfromEmployee")

   #fetchingtherowsfromthecursorobject
   result = cur.fetchall()

   print("Name    id    Salary");
   forrowinresult:
      print("%s    %d    %d"%(row[0],row[1],row[2]))
except:
   myconn.rollback()
myconn.close()
```

Output:

```
Name    id
        SalaryJohn
        101    25000
John    102    25000
David    103    25000
Nick    104    90000
Mike    105    28000
```

Usingwhereclause

We can restrict the result produced by the select statement by using the where clause. This will extract only those columns which satisfy the where condition.

Considerthefollowingexample.

Example:printingthenamesthatstart withj

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
    #ReadingtheEmployeedata
    cur.execute("selectname,id,salaryfromEmployeewherename like'J%'")

    #fetchingtherowsfromthecursorobject
    result = cur.fetchall()

    print("Name    id    Salary");

    forrowinresult:
        print("%s    %d    %d"%(row[0],row[1],row[2]))
except:
    myconn.rollback()

myconn.close()
```

Output:

```
Name    id
        SalaryJohn
        101    25000
John    102    25000
```

Example:printingthenameswith id=101,102,and103

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
    #ReadingtheEmployeedata
    cur.execute("selectname,id,salaryfromEmployeewhere idin(101,102,103)")

    #fetchingtherowsfromthecursorobject
    result = cur.fetchall()

    print("Name    id    Salary");

    forrowinresult:
        print("%s    %d    %d"%(row[0],row[1],row[2]))
except:
    myconn.rollback()

myconn.close()
```

Output:

```
Name    id
        SalaryJohn
        101    25000
John    102    25000
David    103    2500
```

Orderingthe result

TheORDERBYclauseisusedtoordertheresult.Considerthefollowingexample.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
   #ReadingtheEmployeedata
   cur.execute("selectname, id, salaryfromEmployeeorderbyname")

   #fetchingtherowsfromthecursorobject
   result = cur.fetchall()

   print("Name    id    Salary");

   forrowinresult:
      print("%s    %d    %d"%(row[0],row[1],row[2]))
except:
   myconn.rollback()

myconn.close()
```

Output:

```
Name    id
         SalaryDavid
```

103	25000

```
John    101    25000
John    102    25000
Mike     105    28000
Nick    104    90000
```

OrderbyDESC

Thisorderstheresultinthedecreasing orderofaparticularcolumn.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
    #ReadingtheEmployeedata
    cur.execute("selectname,id, salaryfromEmployeeorder bynamedesc")

    #fetchingtherowsfromthecursorobject
    result = cur.fetchall()

    #printing the result
    print("Name    id    Salary");
    for row in result:
        print("%s    %d    %d"%(row[0],row[1],row[2]))

except:
    myconn.rollback()

myconn.close()
```

Output:

```
Name    id    Salary
```

```
Nick    104    90000
Mike    105    28000
John    101    25000
John    102    25000
David    103    25000
```

UpdateOperation

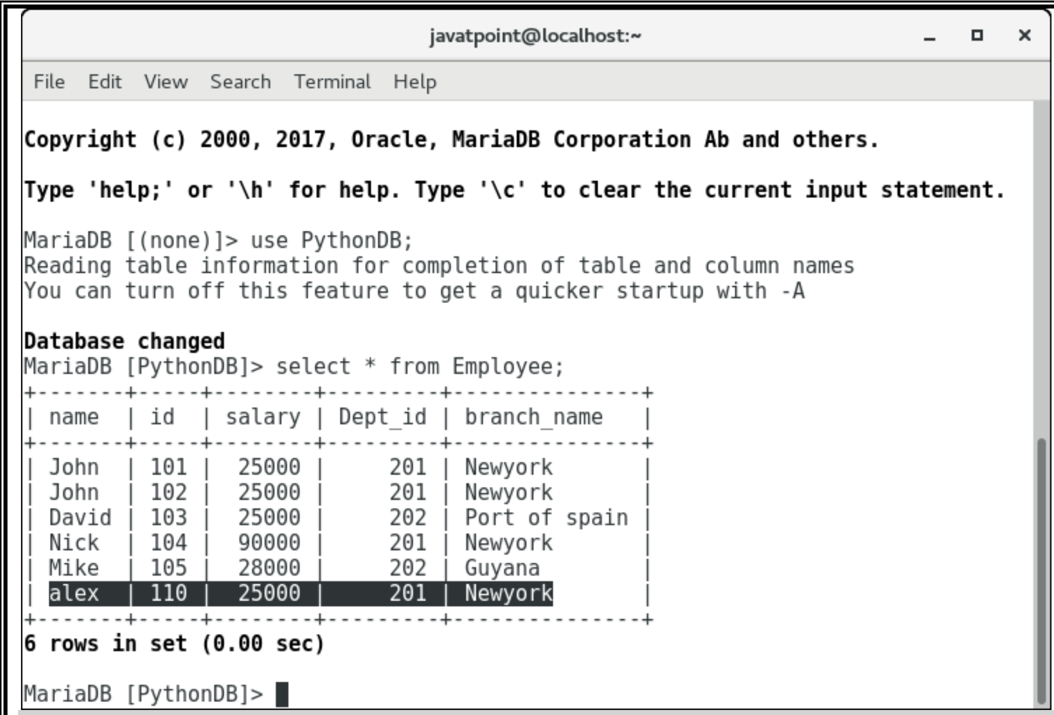
The UPDATE-SET statement is used to update any column inside the table. The following SQL query is used to update a column.

```
>updateEmployeeset name= 'alex'where id=110
```

Consider the following example.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="PythonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
    #updating the name of the employee whose id is 110
    cur.execute("updateEmployeeset name='alex'whereid=110")
    myconn.commit()
except:

    myconn.rollback()

myconn.close()
```



```
Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

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

MariaDB [(none)]> use PythonDB;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MariaDB [PythonDB]> select * from Employee;
+-------+-----+--------+---------+---------------+
| name  | id  | salary | Dept_id | branch_name   |
+-------+-----+--------+---------+---------------+
| John  | 101 |  25000 |     201 | Newyork       |
| John  | 102 |  25000 |     201 | Newyork       |
| David | 103 |  25000 |     202 | Port of spain |
| Nick  | 104 |  90000 |     201 | Newyork       |
| Mike  | 105 |  28000 |     202 | Guyana        |
| alex  | 110 |  25000 |     201 | Newyork       |
+-------+-----+--------+---------+---------------+
6 rows in set (0.00 sec)

MariaDB [PythonDB]>
```

DeleteOperation

TheDELETE FROMstatement isusedto deleteaspecificrecordfromthetable. Here, wemust impose a condition using WHERE clause otherwise all the records from the table will be removed.

The followingSQLqueryisusedtodeletetheemployeedetailwhoseidis110fromthe table.

>delete**from**Employeewhereid=110

Consider the following example.

Example

import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="PythonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:

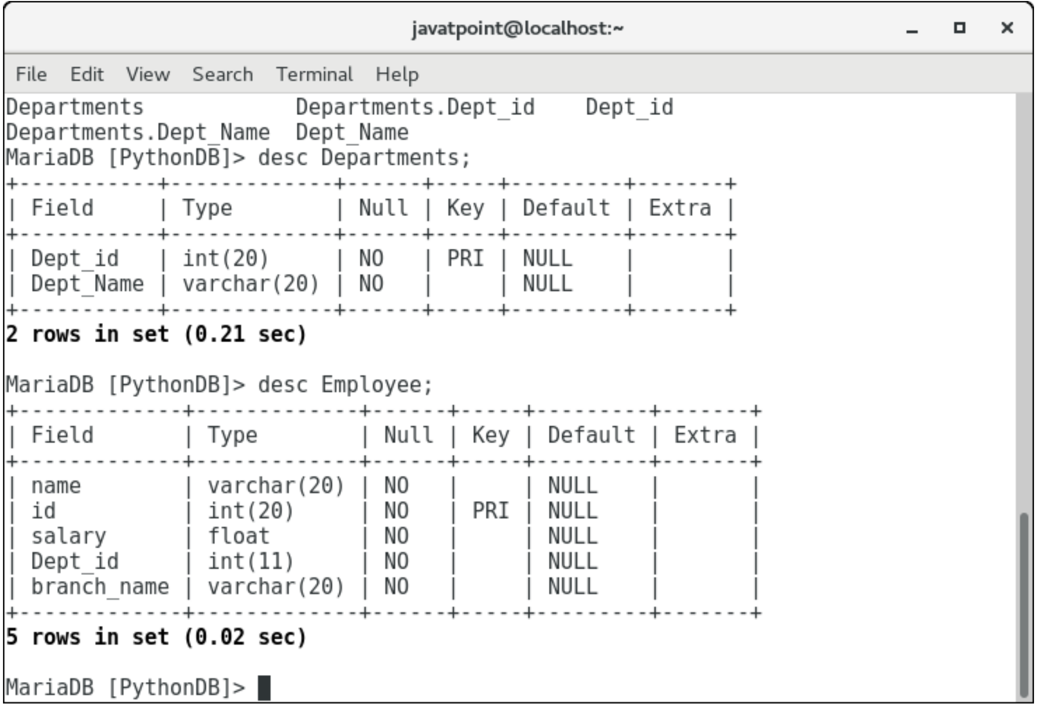
```
    #Deleting the employee details whose id is 110
    cur.execute("deletefromEmployeewhereid=110")
    myconn.commit()
except:

    myconn.rollback()

myconn.close()
```

JoinOperation

We can combine the columns fromtwo or more tables by using some common column among them by using the join statement.

Wehaveonlyonetableinourdatabase,let'screateonemoretableDepartmentswithtwocolumns department_id and department_name.

createtableDepartments(Dept_idint(20)primarykey**not**null,Dept_Namevarchar(20)**not**null);



```
Departments           Departments.Dept_id    Dept_id
Departments.Dept_Name  Dept_Name
MariaDB [PythonDB]> desc Departments;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Dept_id   | int(20)     | NO   | PRI | NULL    |       |
| Dept_Name | varchar(20) | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
2 rows in set (0.21 sec)

MariaDB [PythonDB]> desc Employee;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| name        | varchar(20) | NO   |     | NULL    |       |
| id          | int(20)     | NO   | PRI | NULL    |       |
| salary      | float       | NO   |     | NULL    |       |
| Dept_id     | int(11)     | NO   |     | NULL    |       |
| branch_name | varchar(20) | NO   |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
5 rows in set (0.02 sec)

MariaDB [PythonDB]> 
```

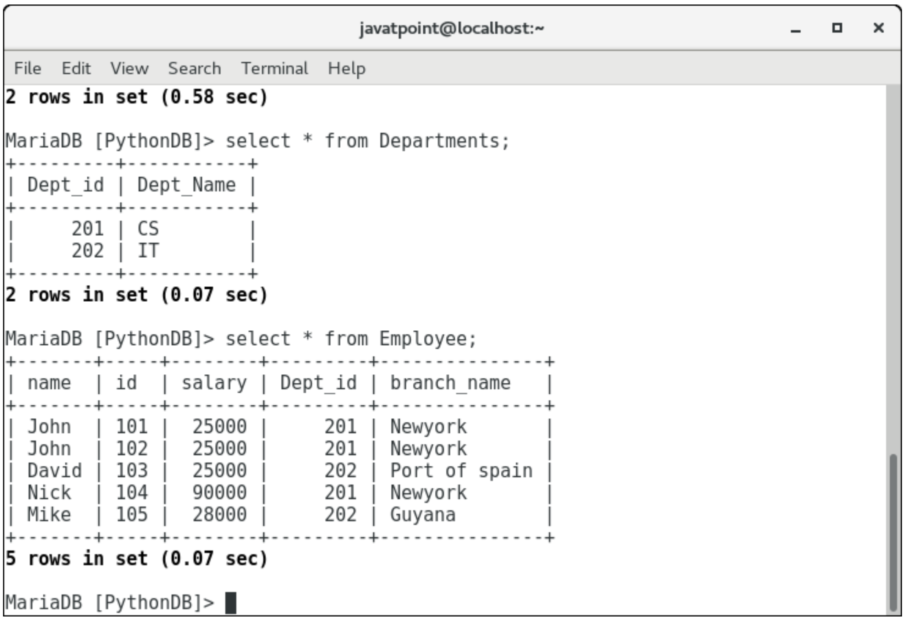
As wehave createdanew table Departments asshownin theaboveimage.However,we haven'tyet inserted any value inside it.

Let'sinsertsomeDepartmentsidsanddepartmentsnamessothatwecanmapthistoourEmployee table.

insertintoDepartmentsvalues(201, "CS");

insertintoDepartmentsvalues(202, "IT");

Let'slookatthevalues insertedineachofthetables. Considerthefollowingimage.



```
2 rows in set (0.58 sec)

MariaDB [PythonDB]> select * from Departments;
+---------+-----------+
| Dept_id | Dept_Name |
+---------+-----------+
|     201 | CS        |
|     202 | IT        |
+---------+-----------+
2 rows in set (0.07 sec)

MariaDB [PythonDB]> select * from Employee;
+-------+-----+--------+---------+---------------+
| name  | id  | salary | Dept_id | branch_name   |
+-------+-----+--------+---------+---------------+
| John  | 101 |  25000 |     201 | Newyork       |
| John  | 102 |  25000 |     201 | Newyork       |
| David | 103 |  25000 |     202 | Port of spain |
| Nick  | 104 |  90000 |     201 | Newyork       |
| Mike  | 105 |  28000 |     202 | Guyana        |
+-------+-----+--------+---------+---------------+
5 rows in set (0.07 sec)

MariaDB [PythonDB]> 
```

Now,let'screateapythonscript thatjoinsthetwo tablesonthecommoncolumn,i.e.,dept_id.

Example

```
importmysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")
  #creatingthecursorobject
cur = myconn.cursor()
  try:
    #joining thetwotablesondepartments_id
    cur.execute("selectEmployee.id,Employee.name,Employee.salary,Departments.Dept_id,Departme
nts.Dept_Name from Departments join Employee on Departments.Dept_id = Employee.Dept_id")
    print("ID    Name    Salary    Dept_Id    Dept_Name")
    forrowincur:
        print("%d    %s    %d    %d    %s"%(row[0],row[1],row[2],row[3],row[4]))

except:
    myconn.rollback()

myconn.close()
```

Output:

```
ID   Name    Salary   Dept_Id   Dept_Name
101John      25000    201            CS
102John      25000    201            CS
103David25000          202                IT
104Nick      90000    201            CS
105Mike       28000   202            IT
```

RightJoin

Right join shows all the columns of the right-hand side table as we have two tables in the database PythonDB, i.e., Departments and Employee. We do not have any Employee in the table who is not working for any department (Employee for which department id is null). However, to understand the concept of right join let's create the one.

ExecutethefollowingqueryontheMySQL server.

```
insert into Employee(name, id, salary, branch_name) values ("Alex",108,29900,"Mumbai");
```

Thiswillinsert anemployeeAlexwhodoesn't workforanydepartment (department idisnull).

Now,wehaveanemployeeintheEmployeetablewhosedepartmentidisnotpresentinthe Departments table. Let's perform the right join on the two tables now.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
   #joining thetwotablesondepartments_id
   result = cur.execute("select Employee.id, Employee.name, Employee.salary, Departments.Dept_id,
Departments.Dept_Name fromDepartmentsright joinEmployeeonDepartments.Dept_id=Employee.
Dept_id")
```

```
    print("ID    Name    Salary    Dept_Id    Dept_Name")

    forrowincur:
        print(row[0],"    ", row[1],"    ",row[2],"    ",row[3],"    ",row[4])



except:
    myconn.rollback()

myconn.close()
```

Output:

```
ID    Name    Salary    Dept_Id    Dept_Name
101      John      25000.0      201      CS
102      John      25000.0      201      CS
103      David      25000.0      202      IT
104      Nick      90000.0      201      CS
105      Mike      28000.0      202      IT
108      Alex      29900.0      None      None
```

Left Join

The left joincoversallthe data fromthe left-hand side table. It has just opposite effect to the right join. Consider the following example.

Example

```
import mysql.connector

#Createtheconnectionobject
myconn= mysql.connector.connect(host ="localhost",user="root",passwd="google",database="Py
thonDB")

#creatingthecursorobject
cur = myconn.cursor()

try:
    #joining thetwotablesondepartments_id
    result = cur.execute("select Employee.id, Employee.name, Employee.salary, Departments.Dept_id,
Departments.Dept_Name fromDepartmentsleft joinEmployeeonDepartments.Dept_id=Employee.D
ept_id")
    print("ID    Name    Salary    Dept_Id    Dept_Name")
    forrowincur:
        print(row[0],"    ", row[1],"    ",row[2],"    ",row[3],"    ",row[4])




except:
    myconn.rollback()

myconn.close()
```

Output:

```
ID    Name    Salary    Dept_Id    Dept_Name
101      John      25000.0      201      CS
102      John      25000.0      201      CS
103      David      25000.0      202      IT
104      Nick      90000.0      201      CS
105      Mike      28000.0      202      IT
```

Transactionproperty

Thetransactionhasthefourproperties.Theseareusedtomaintainconsistencyinadatabase,before and after the transaction.

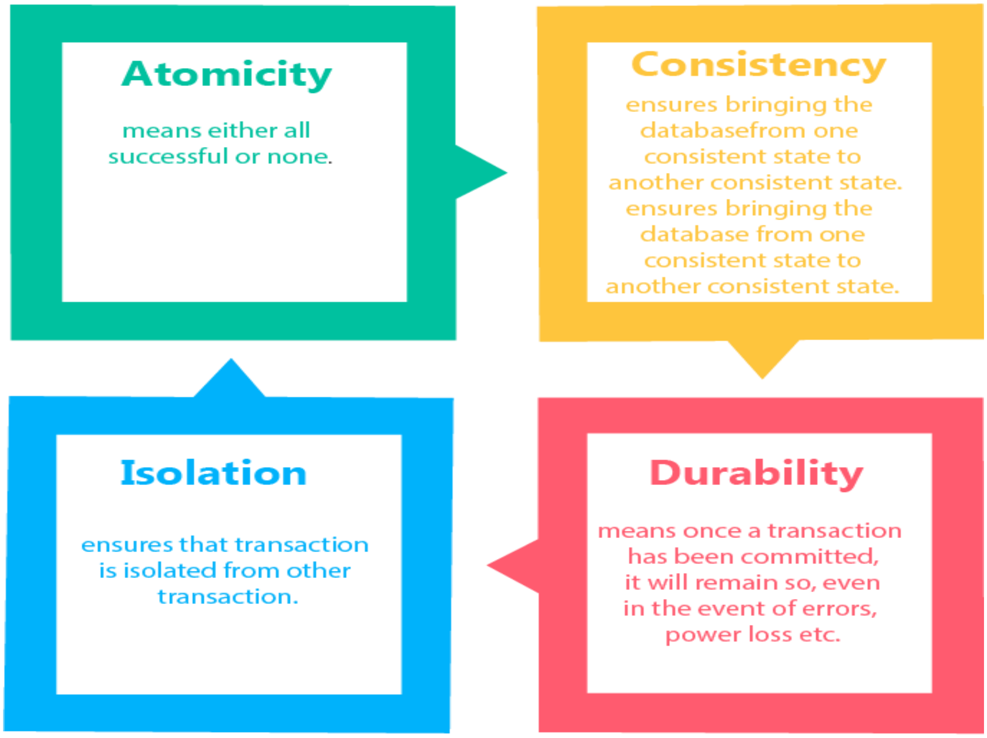
PropertyofTransaction

Atomicity

Consistency

Isolation

Durability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

Itstatesthatalloperationsofthetransactiontakeplaceatonceifnot,thetransaction is aborted.

There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicityinvolvesthefollowingtwooperations:

Abort: Ifatransactionabortsthenallthechangesmade arenotvisible.

Commit: Ifatransactioncommitsthenallthechangesmadeare visible.

Example: Let's assume that following transaction T consisting ofT1 and T2. A consists ofRs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:=A-100
Write(A)
	Read(B)
	Y:=Y+100
	Write(B)

Aftercompletionofthe transaction,AconsistsofRs500 and BconsistsofRs400.

Ifthe transactionT fails after the completionoftransactionT1 but before completionoftransactionT2, thentheamount willbedeductedfromAbut not addedtoB.Thisshowsthe inconsistent databasestate. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

Theintegrityconstraintsaremaintainedsothatthedatabaseisconsistentbeforeandafterthe transaction.

Theexecutionofatransactionwillleaveadatabase ineither itspriorstablestateoranewstablestate. The

consistent property of database states that every transaction sees a consistent database instance.

Thetransactionisusedtotransformthedatabasefromoneconsistentstatetoanother consistentstate.

Forexample: Thetotalamountmustbemaintainedbeforeorafterthetransaction.

TotalbeforeToccurs=600+300=900
Total after T occurs= 500+400=900

Therefore, the database is consistent. Inthe case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

Itshowsthatthedatawhichisusedatthetimeofexecutionofatransactioncannotbeusedbythe second transaction until the first one is completed.

In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.

TheconcurrencycontrolsubsystemoftheDBMSenforced theisolationproperty.

Durability

The durability property is usedtoindicate the performance of the database's consistentstate. Itstates that the transaction made the permanent changes.

They cannot be lost by the erroneous operation ofa faulty transaction or by the system failure. When a transactionis completed, thenthedatabasereaches astateknownastheconsistent state. That consistent state cannot be lost, even in the event of a system's failure.

TherecoverysubsystemoftheDBMShas the responsibilityofDurabilityproperty.

MySQLRegularExpressions

Aregularexpressionisaspecialstringthatdescribesasearchpattern.It'sapowerfultooltogive a**conciseandflexiblewayforidentifying textstrings**suchascharactersandwordsbased onpatterns.

It uses its own syntax that can be interpreted bya regular expressionprocessor. A regular expression is widely used in almost all platforms, from programming languages to databases, including MySQL.

Aregular expressionusesthe backslashasan **escape character**that should beconsidered inthepattern match if double backslashes have used. The regular expressions are not case sensitive. **Itis abbreviated as REGEX or REGEXP in MySQL**.

The advantage of using regular expression is that we are not limited to search for a string based on a fixed patternwiththe percent (%)signand underscore(_) inthe LIKEoperator.The regular expression has more meta-characters that allow more flexibility and control while performing pattern matching.

We have**previously learned about wildcards**, which allows us to get a similar result as regular expressions. So we may ask **why we learn regular expressions** if we will get the same result as the wildcards. It is because regular expressions allow us to search data matching even more complex ways compared to wildcards.

Syntax

MySQL adapts the regular expression implemented by**Henry Spencer**. MySQLallows us to match patterns right in the SQL statements by using the REGEXP operator. The following is the basic syntax that illustrates the use of regular expressions in MySQL:

SELECTcolumn_lists**FROM**table_name**WHERE** field_nameREGEXP'pattern';

In this syntax, the**column_list** indicates the column name returns in the result set. The**table_name** is the name ofthe table that data will be retrieved using the pattern. The **WHEREfield_name**represents the column name onwhichthe regular expressionis performed. The REGEXPis the regular expression operator,andthe **pattern** isthesearchconditiontobematchedbyREGEXP.Wecanalsouse the**RLIKE**operator, which is the synonym for REGEXP that gives the same results as REGEXP. We can avoid the confusion to use this statement with the LIKE operator by using the REGEXP instead of LIKE.

This statement returns**true** if a value in the WHERE field_name matches the pattern. Otherwise, it returns**false**. Ifeither field_nameorpatternis **NULL**, theresult isalwaysNULL. The negationformof the REGEXP operator is NOT REGEXP.

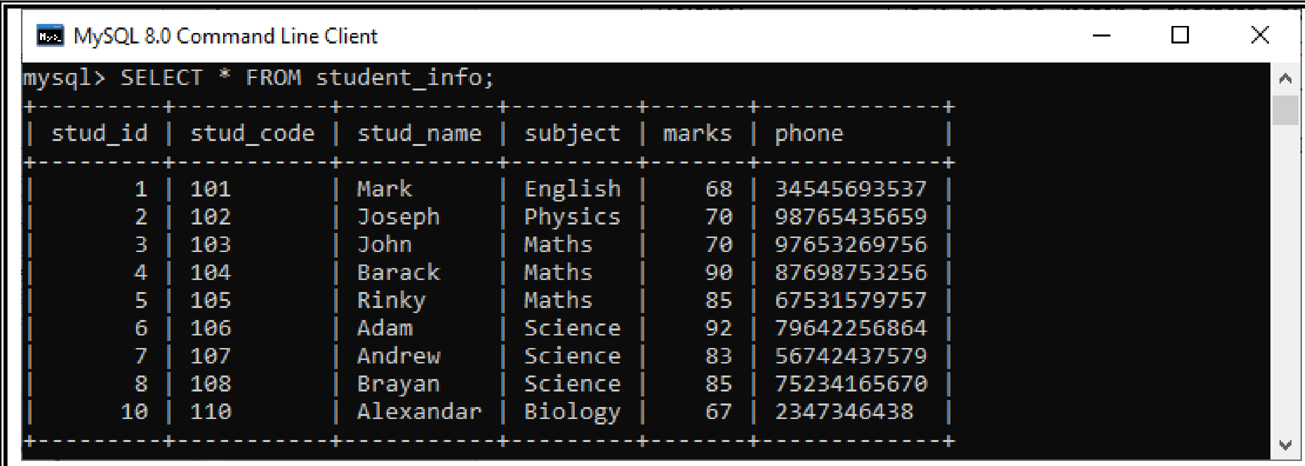
RegularExpressionMeta-Characters

Thefollowingtableshowsthemostcommonlyusedmeta-charactersandconstructsinaregular expression:

Meta-Character	Descriptions
^	Thecaret(^)characterisusedtostartmatchesatthebeginningofasearchedstring.
$	Thedollar($) character isusedtostartmatchesattheendofasearchedstring.
.	Thedot(.) character matchesanysinglecharacterexceptfor anewline.
[abc]	Itisused to matchanycharactersenclosedinthesquarebrackets.
[^abc]	Itisusedtomatchanycharactersnotspecifiedinthesquarebrackets.
*	Theasterisk(*)charactermatcheszero(0)or moreinstancesoftheprecedingstrings .
+	Theplus(+) character matchesoneor moreinstancesofprecedingstrings.
{n}	Itisused to matchn instancesofthepreceding element.
{m,n}	Itisusedto match mtoninstanceofthepreceding element.
p1|p2	Itisusedto isolate alternativesthatmatchanyofthepatternsp1 orp2.
?	Thequestionmark(?)charactermatcheszero(0)oroneinstanceofprecedingstrings.
[A-Z]	Itisused tomatchanyuppercasecharacter.
[a-z]	Itisusedtomatchanylower case character.
[0-9]	Itisused to matchnumericdigitsfrom0 to 9.
[[:<:]]	Itmatchesthebeginning ofwords.
[[:>:]]	Itmatchestheend ofwords.
[:class:]	It is used to match a character class, i.e. [:alpha:]matches letters, [:space:] es matchwhite space, [:punct:] matches punctuations and [:upper:] for upper-class letters.

Letusunderstandtheregularexpressionsusingpracticalexamplesgivenbelow:

Supposewehaveatablenamed **student_info**thatcontainsthefollowingdata.Wewilldemonstrate various examples based on this table data.



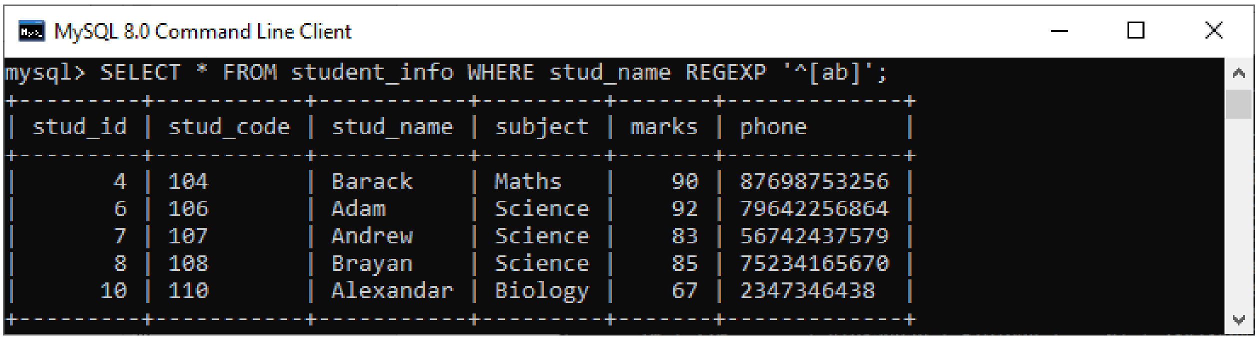
MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM student_info;
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       1 | 101       | Mark      | English |    68 | 34545693537 |
|       2 | 102       | Joseph    | Physics |    70 | 98765435659 |
|       3 | 103       | John      | Maths   |    70 | 97653269756 |
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
|       5 | 105       | Rinky     | Maths   |    85 | 67531579757 |
|       6 | 106       | Adam      | Science |    92 | 79642256864 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
|      10 | 110       | Alexandar | Biology |    67 | 2347346438  |
+---------+-----------+-----------+---------+-------+-------------+
```

If we want to **search for students whose name start with "A or B"**, we can use a regular expression together with the meta-characters as follows:

mysql>**SELECT*FROM**student_info **WHERE**stud_nameREGEXP'^[ab]';

Executing the statement, we will get the desired result. See the below output:



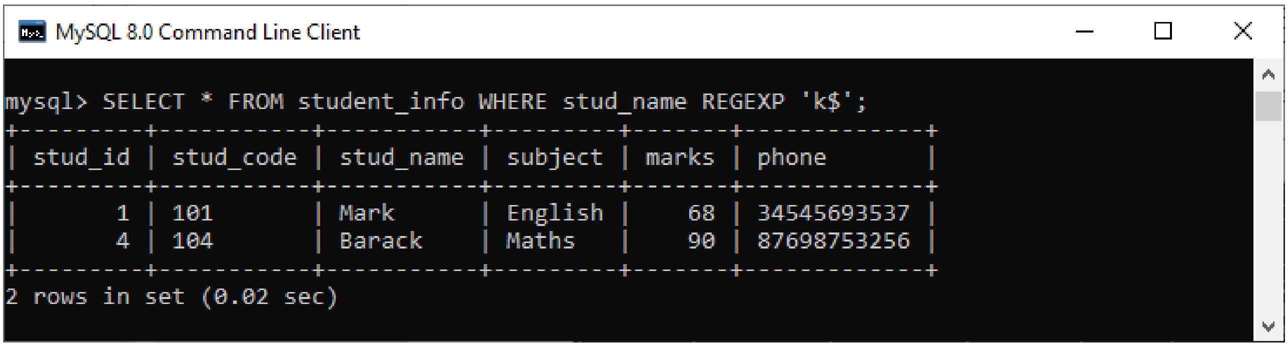
MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM student_info WHERE stud_name REGEXP '^[ab]';
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
|       6 | 106       | Adam      | Science |    92 | 79642256864 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
|      10 | 110       | Alexandar | Biology |    67 | 2347346438  |
+---------+-----------+-----------+---------+-------+-------------+
```

Ifwewantto **get the student information whose name ends with k**, wecanuse'k$'meta-characterto match the end of a string as follows:

mysql>**SELECT*FROM**student_info **WHERE**stud_nameREGEXP'k$';

Executing the statement, we willget the desired result. Seethe below output:



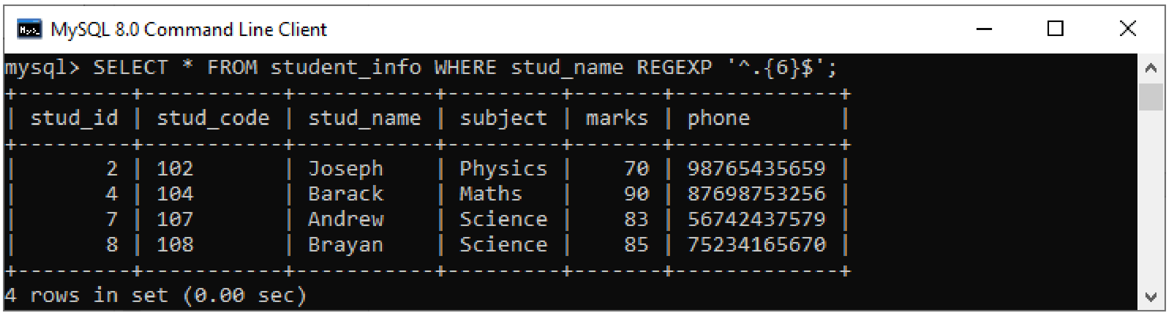
MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM student_info WHERE stud_name REGEXP 'k$';
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       1 | 101       | Mark      | English |    68 | 34545693537 |
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
+---------+-----------+-----------+---------+-------+-------------+
2 rows in set (0.02 sec)
```

If we want to **get the student information whose name contains exactly six characters**, we can do this using '^'and'**$ meta-characters**. These characters match the **beginning and end**of the student name and repeat {6} times of anycharacter '**.**' in-between as shown in the following statement:

mysql>**SELECT*FROM**student_info **WHERE**stud_nameREGEXP'^.{6}$';

Executing the statement, we will get the desired result. See the below output:



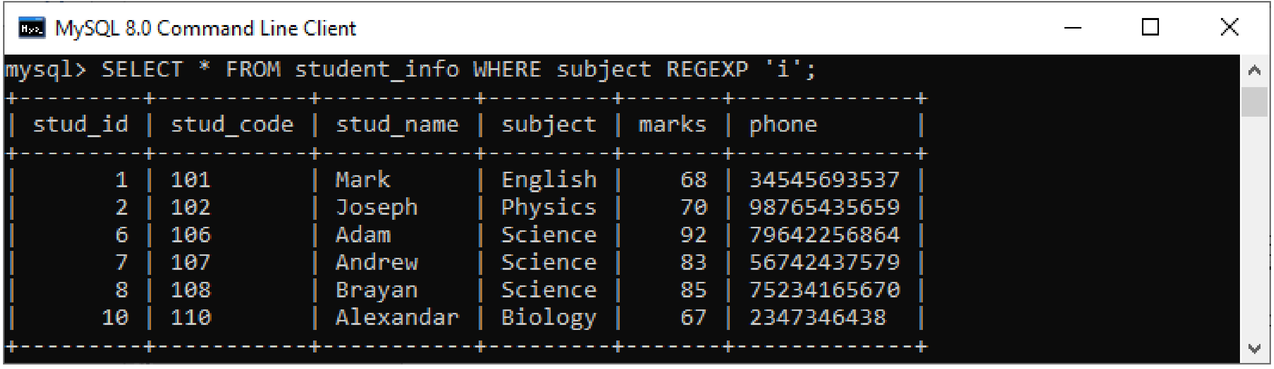
```
mysql> SELECT * FROM student_info WHERE stud_name REGEXP '^.{6}$';
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       2 | 102       | Joseph    | Physics |    70 | 98765435659 |
|       4 | 104       | Barack    | Maths   |    90 | 87698753256 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
+---------+-----------+-----------+---------+-------+-------------+
4 rows in set (0.00 sec)
```

Ifwewant to **get the student info whose subjects contains 'i' characters**, wecando this byusing the below query:

mysql>**SELECT** * **FROM** student_info **WHERE** subject REGEXP 'i';

Executingthestatement, wewillgetthedesiredresult. Seethebelowoutput:

Regular Expression Functions and Operators



```
mysql> SELECT * FROM student_info WHERE subject REGEXP 'i';
+---------+-----------+-----------+---------+-------+-------------+
| stud_id | stud_code | stud_name | subject | marks | phone       |
+---------+-----------+-----------+---------+-------+-------------+
|       1 | 101       | Mark      | English |    68 | 34545693537 |
|       2 | 102       | Joseph    | Physics |    70 | 98765435659 |
|       6 | 106       | Adam      | Science |    92 | 79642256864 |
|       7 | 107       | Andrew    | Science |    83 | 56742437579 |
|       8 | 108       | Brayan    | Science |    85 | 75234165670 |
|      10 | 110       | Alexandar | Biology |    67 | 2347346438  |
+---------+-----------+-----------+---------+-------+-------------+
```

Thefollowingarethelistofregularfunctionsandoperatorsin MySQL:

Name	Descriptions
NOT_REGEXP	Itis thenegationofaREGEXP operator.

REGEXP	This operator represents whether the string matches regular expression or not.
RLIKE	This operator represents whether the string matches regular expression or not.
REGEXP_INSTR()	It is a function that gives a result when the starting index of substringmatches a regular expression.
REGEXP_LIKE()	This function represents whether the string matches regular expression or not.
REGEXP_REPLACE()	Itgivesresultsbyreplacingsubstringsthatmatchtheregular expression.
REGEXP_SUBSTRING()	Thisfunctionreturnsubstringthatmatchesaregular expression.

Let's seeallofthemin detail.

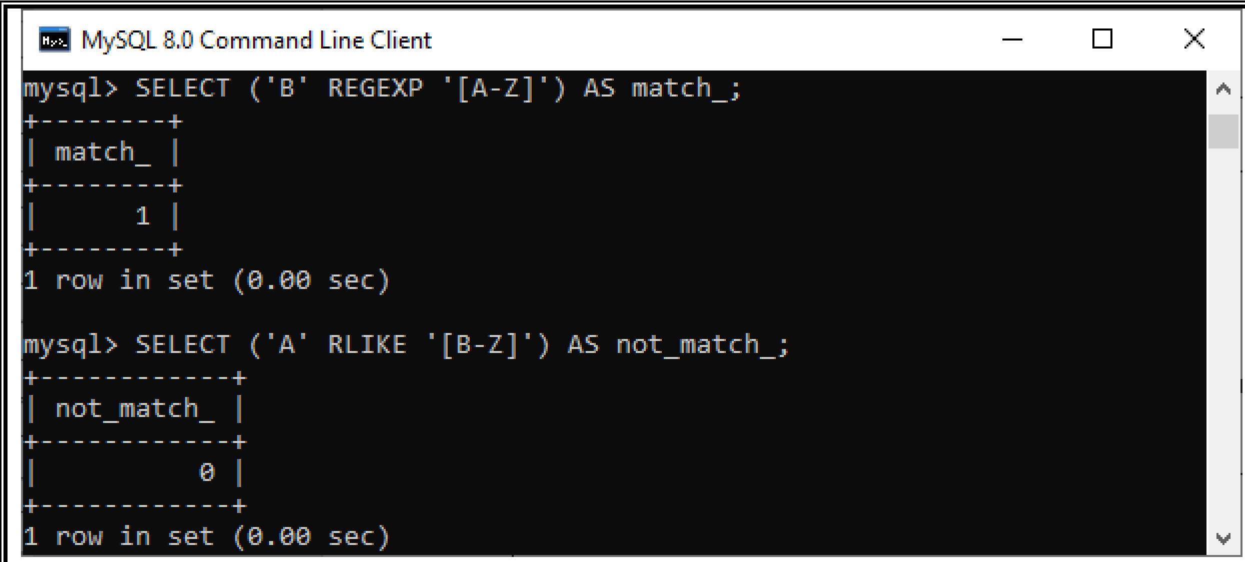
REGEXP,RLIKE, &REGEXP_LIKE()

Although these functions and operators return the same result, **REGEXP_LIKE**() gives us more functionality with the optional parameters. We can use them as follows:

expressionREGEXPpattern expression
RLIKE pattern
REGEXP(expression, pattern[,match_type])

These statements give output whether string expression matches regular expression pattern or not. We willget 1 ifanexpression matches the pattern. Otherwise, theyreturn0. The below examples explain it more clearly.

Inthe below image,the first statement returns'1'because**'B'**is inthe range A-Z. The second statement limited the range of the pattern to B-Z. So **'A'**will not match any character within the range, and MySQL returns 0. Here we have used the alias **match_ and not_match_** so that the returned column will be more understandable.



```
mysql> SELECT ('B' REGEXP '[A-Z]') AS match_;
+--------+
| match_ |
+--------+
|      1 |
+--------+
1 row in set (0.00 sec)

mysql> SELECT ('A' RLIKE '[B-Z]') AS not_match_;
+------------+
| not_match_ |
+------------+
|          0 |
+------------+
1 row in set (0.00 sec)
```

REGEXP_LIKE()Parameter

Thefollowingarethe **fivepossibleparameters**tomodifythefunctionoutput:

c:Itrepresentsacase-sensitivematching.

i:Itrepresentsacase-insensitivematching.

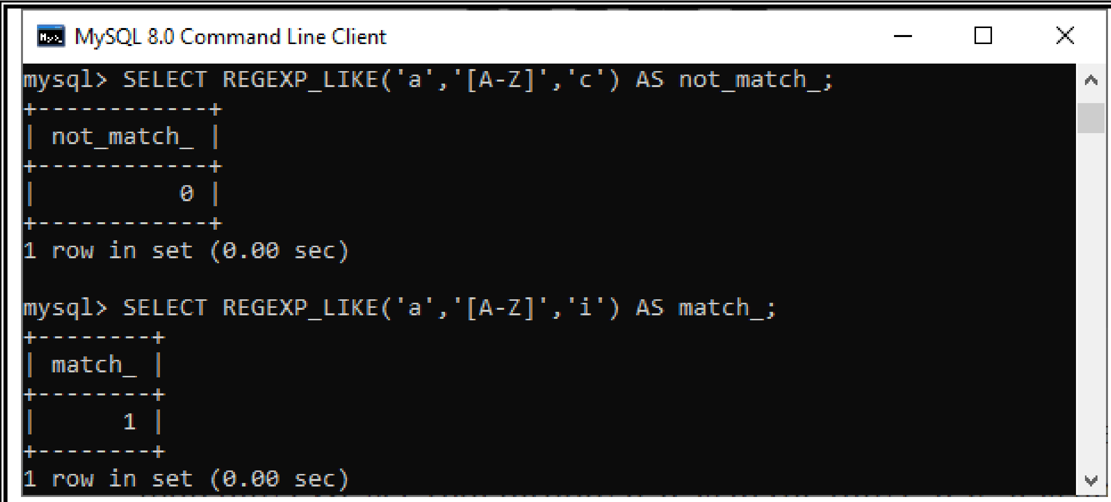
m: It represents a multiple-line mode that allows line terminators within the string. By default, this function matches line terminators at the start and end of the string.

n:Itisusedtomodifythe. (dot)charactertomatchline terminators.

u:ItrepresentsUnix-onlyline endings.

Example

Inthisexample,wehaveaddedthe **'c'and'i'**asanoptionalparameter,whichinvokes **case- sensitive**and**case-insensitive** matching. The first querygivestheoutput 0because 'a' is inthe range 'a- z', but not in the range of capital letters A-Z. The second query gives the output 1 because of case- insensitive features.



```
mysql> SELECT REGEXP_LIKE('a','[A-Z]','c') AS not_match_;
+------------+
| not_match_ |
+------------+
|          0 |
+------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_LIKE('a','[A-Z]','i') AS match_;
+--------+
| match_ |
+--------+
|      1 |
+--------+
1 row in set (0.00 sec)
```

NOTREGEXP&NOTRLIKE

They are regular expression operators that compare the specified pattern and return the result, which doesnot matchthepatterns. Theseoperatorsreturn1 ifno matchis found. Otherwise, theyreturn0. We can use these functions as follows:

SELECT(exprNOTREGEXPpat); OR

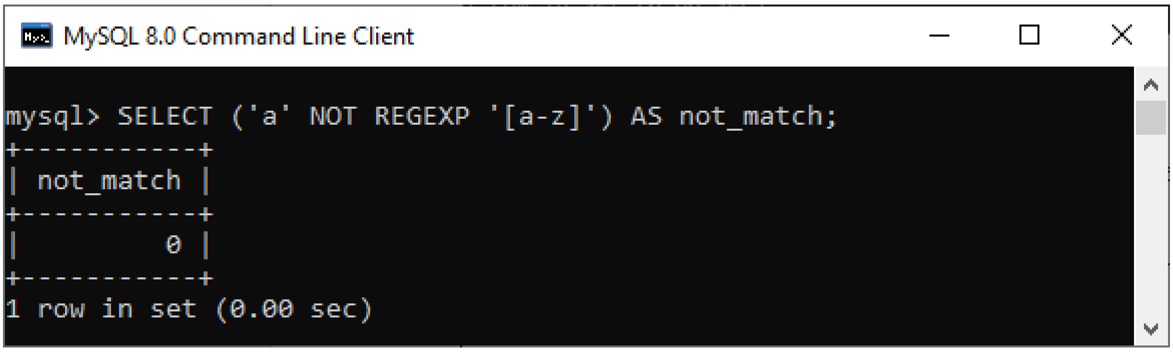
SELECT(exprNOTRLIKE pat);

Example

Thebelowstatementreturns0because**'a'**isfoundinthegivenrange.

mysql>**SELECT**('a'NOTREGEXP'[a-z]')**AS**not_match; Here is

the output:



```
mysql> SELECT ('a' NOT REGEXP '[a-z]') AS not_match;
+-----------+
| not_match |
+-----------+
|         0 |
+-----------+
1 row in set (0.00 sec)
```

REGEXP_INSTR()

It is a functionthat gives a result whenthe starting indexofsubstring expression matches the pattern. It returns 0 if there is no match found. If either expression or pattern is **NULL**, it returns NULL. Here indexing starts at 1.

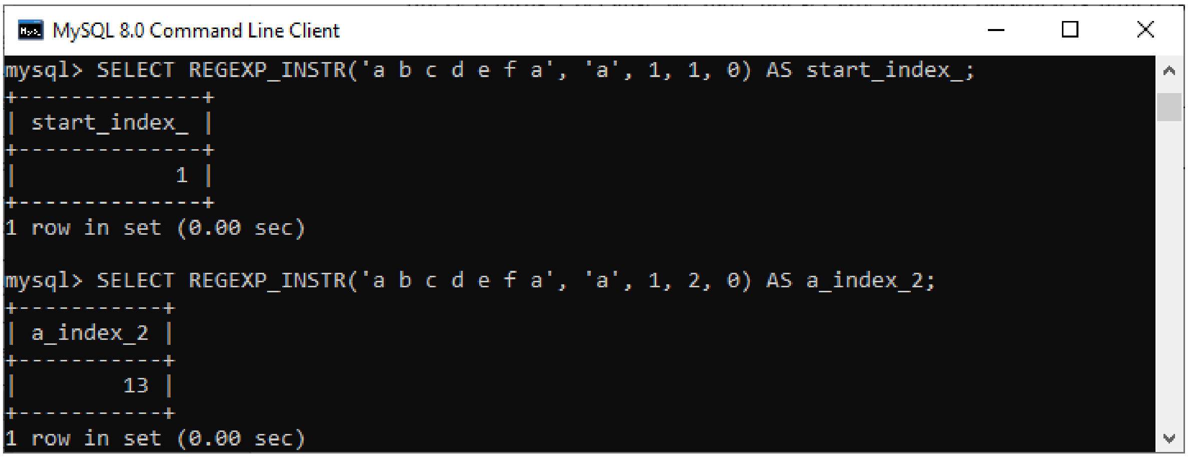
REGEXP_INSTR(expr,pat[,pos[,occurrence[,return_option[,match_type]]]])

Thisfunctionusesvariousoptionalparametersthatarepos,occurrence, return_option,match_type, etc.

Example

Suppose we want to get the index position of substring 'a' within expr (a b c d e f a). The first query returns 1 because we have not set anyoptionalparameters, which is the string's first index. The second query returns 13 because we have modified the query with optional parameter occurrence.

SELECTREGEXP_INSTR('abcd efa','a',1,1,0)**AS**start_index_;
SELECTREGEXP_INSTR('abcd efa','a',1,2,0)**AS**a_index_2;



```
mysql> SELECT REGEXP_INSTR('a b c d e f a', 'a', 1, 1, 0) AS start_index_;
+--------------+
| start_index_ |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('a b c d e f a', 'a', 1, 2, 0) AS a_index_2;
+-----------+
| a_index_2 |
+-----------+
|        13 |
+-----------+
1 row in set (0.00 sec)
```

REGEXP_REPLACE()

This function replaces the specified string character by matching characters and then returns the resulting string. Ifanyexpression, pattern, or replaceable string is not found, it willreturn NULL. This function can be used as follows:

SELECTREGEXP_REPLACE('expression','character','replace_character');

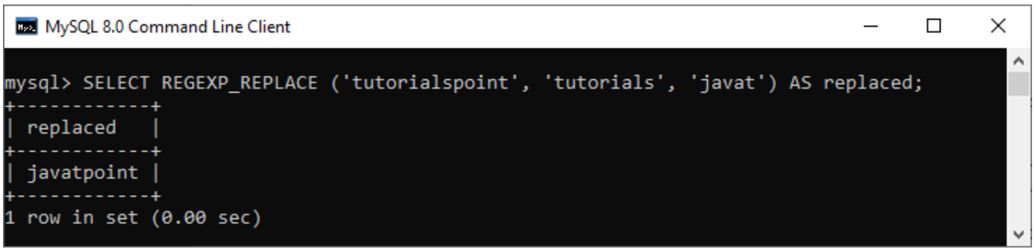
Thereplacecharacter usestheoptionalparameterssuchaspos,occurrence,andmatch_type.

Example

Thisstatement replacesthe'tutorials'patternwiththe'javat'pattern.

mysql>**SELECT**REGEXP_REPLACE('tutorialspoint','tutorials','javat')**AS**replaced;

Here is the output:



```
mysql> SELECT REGEXP_REPLACE ('tutorialspoint', 'tutorials', 'javat') AS replaced;
+------------+
| replaced   |
+------------+
| javatpoint |
+------------+
1 row in set (0.00 sec)
```

REGEXP_SUBSTRING()

This functionreturns the substring ofanexpressionthat matches the specified pattern. Ifthe expression or specified patternor even no match is found, it returns NULL. This function can be used as follows:

SELECTREGEXP_SUBSTR('expr','pattern');

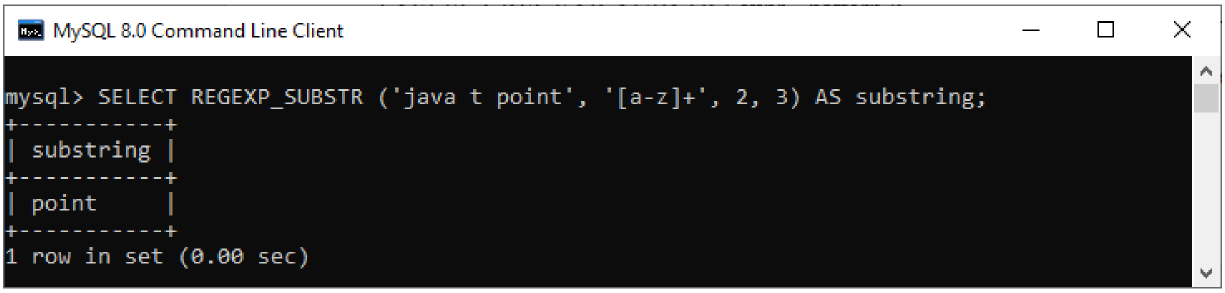
Thepatternusestheoptionalparameterssuchaspos,occurrence,andmatch_type.

Example

Thisstatementreturnsthe'point'pattern, whichisthethirdoccurrenceofthegivenrange.

mysql>**SELECT**REGEXP_SUBSTR('javatpoint','[a-z]+',2,3)**AS**substring; Here is

the output:



```
mysql> SELECT REGEXP_SUBSTR ('java t point', '[a-z]+', 2, 3) AS substring;
+-----------+
| substring |
+-----------+
| point     |
+-----------+
1 row in set (0.00 sec)
```

<u>DATABASEADAPTER:</u>

Python offers database adapters through its modules that allow access to major databases such as MySQL, PostgreSQL, SQL Server, and SQLite. Furthermore, all of these modules rely on Python's database API (DB-API) for managing databases.

Database adapter

Atualizadopelaúltimavez:2023-01-26

Out-of-the-box, virtual member manager provides a default database profile repository (wimDB), that supports all common virtual member manager supported profile repository features.

Note: Thedatabaseuserwhoconfigurestherepositoryneedstohavedatabaseadministratorprivileges suchaspermissionstocreatetables inthedatabase schemaandto accessthedatainthedatabasetables. The database repository is designed using relationaldatabase. The database adapter is a bridge between the virtual member manager profile and schema managers and the underlying database. The adapter looks up the data source and updates or queries the database using SQL queries.

The database repository supports all entity types that are predefined in the virtual member manager model schema definition, such as: Person, Group, OrgContainer, and PersonAccount. The database adapter can also support any user-defined entity types that extend from the virtual member manager standard schema. It creates the user-defined entities in the database during runtime.

The database repository supports predefined property definitions that are consistent with the virtual member manager schema, as well as dynamically defined new properties during runtime.

A database repository property definition extends the virtual member manager schema property definition. It contains:

name

Specifiesthenameoftheproperty.Thisisarequiredproperty.

datatype

Specifies a data type. String, Integer, Long, Double, Timestamp, Base64Binary, Identifier and Object are the default supported data types. If a property has user-defined data type, set Objects as data type andsettheuser-defineddatatypeclassnameintheDBPROPtable“classname”column.For example, to support a Boolean data type, set the “type_id” column OBJECT and set the “classname” column to java.lang.Boolean. This is a required property.

Note: SupporteddatatypesaredefinedintheSchemaConstant.javafile.

applicableforentitytypes

Specifies a list of entity types for which this property is applicable, for example,PersonAccount;Group. This is a required property.

requiredforentitytypes

Specifiesalistof entity types thatrequire this property value to besetduring the entity creation.This is an optional property.

multiValued

Specifieswhetherthedatabaserepositorycanstoremultiplevaluesforaproperty.Bydefault, multiValued is true. This is an optional parameter.

metaName

Specifiesthenameofmetadata.Bydefault,itissettoDEFAULT,whichmeansthatthereisno associated metadata. This is an optional parameter.

readOnly

Specifiesifapropertyisreadonly.By default,itisfalse.Thisisan optional parameter.

caseExactMatch

Specifiesifapropertyiscasesensitiveduringthesearch.Bydefault,itissettotrue.Thisisan optional parameter.

valueLength

Specifieshemaximumlengthofapropertyifitis Stringtype.Thedefaultvalueis1500.Forother data types, this property is ignored. This is an optional parameter.

isComposite

Specifiesifapropertyisacompositeproperty.Bydefault,itisfalse.Thisisan optionalparameter.

classname