**Python:**

------------------------------

Python is a simple, general purpose, high level, and

object-oriented programming language.

Python is an interpreted scripting language also.

Guido Van Rossum is known as the founder of Python

programming.

Python is easy to learn yet powerful and versatile

scripting language, which makes it attractive for

Application Development.

----------------------------------------------

**Python Features**

Python provides many useful features which make it

popular and valuable from the other programming

languages. It supports object-oriented programming,

procedural programming approaches and provides dynamic

memory allocation. We have listed below a few essential

features.

**1) Easy to Learn and Use**

Python is easy to learn as compared to other programming

languages. Its syntax is straightforward and much the

same as the English language. There is no use of the

semicolon or curly-bracket, the indentation defines

the code block. It is the recommended programming

language for beginners.

**2) Expressive Language**

Python can perform complex tasks using a few lines of

code. A simple example, the hello world program you

simply type print ("Hello World"). It will take

only one line to execute, while Java or C takes

multiple lines.

**3) Interpreted Language**

Python is an interpreted language; it means the Python

program is executed one line at a time. The advantage

of being interpreted language, it makes debugging easy

and portable.

**4) Cross-platform Language**

Python can run equally on different platforms such as

Windows, Linux, UNIX, and Macintosh, etc. So, we can

say that Python is a portable language. It enables

programmers to develop the software for several

competing platforms by writing a program only once.

**5) Free and Open Source**

Python is freely available for everyone. It is freely

available on its official website www.python.org. It

has a large community across the world that is

dedicatedly working towards make new python modules

and functions. Anyone can contribute to the Python

community. The open-source means, "Anyone can download

its source code without paying any penny."

**6) Object-Oriented Language**

Python supports object-oriented language and concepts

of classes and objects come into existence. It supports

inheritance, polymorphism, and encapsulation, etc. The

object-oriented procedure helps to programmer to write

reusable code and develop applications in less code.

**7) Extensible**

It implies that other languages such as C/C++ can be used to compile the

code and thus it can be used further in our Python code.

It converts the program into byte code, and any platform

can use that byte code.

**8) Large Standard Library**

It provides a vast range of libraries for the various

fields such as machine learning, web developer, and

also for the scripting. There are various machine

learning libraries, such as Tensor flow, Pandas, Numpy,

Keras, and Pytorch, etc. Django, flask, pyramids are

the popular framework for Python web development.

**9) GUI Programming Support**

Graphical User Interface is used for the developing

Desktop application. PyQT5, Tkinter, Kivy are the

libraries which are used for developing the web

application.

**10) Integrated**

It can be easily integrated with languages like C,

C++, and JAVA, etc. Python runs code line by line like

C,C++ Java. It makes easy to debug the code.

**Python History and Versions:**

Python laid its foundation in the late 1980s.

The implementation of Python was started in December

1989 by Guido Van Rossum at CWI in Netherland. In

February 1991, Guido Van Rossum published the code

(labeled version 0.9.0) to alt.sources. In 1994,

Python 1.0 was released with new features like lambda,

map, filter, and reduce. Python 2.0 added new features

such as list comprehensions, garbage collection systems.

On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed

to rectify the fundamental flaw of the language.

ABC programming language is said to be the predecessor of Python language,

which was capable of Exception Handling

and interfacing with the Amoeba Operating System.

The following programming languages influence Python:

ABC language.

Modula-3

**Why the Name Python?**

There is a fact behind choosing the name Python. Guido

van Rossum was reading the script of a popular BBC

comedy series "Monty Python's Flying Circus".

It was late on-air 1970s.

Van Rossum wanted to select a name which unique, sort,

and little-bit mysterious. So he decided to select

naming Python after the "Monty Python's Flying Circus"

for their newly created programming language.

Tips to Keep in Mind While Learning Python

The most common question asked by the beginners -

"What is the best way to learn Python"? It is the

initial and relevant question because first step in

learning any programminglanguage is to know how to learn.

The proper way of learning will help us to learn fast

and become a good Python developer.

In this section, we will discuss various tips that we

should keep in mind while learning Python.

1. Make it Clear Why We Want to Learn

The goal should be clear before learning the Python.

Python is an easy, a vast language as well. It includes

numbers of libraries, modules, in-built functions

and data structures. If the goal is unclear then it

will be a boring and monotonous journey of learning

Python. Without any clear goal, you perhaps

won't make it done.

So, first figure out the motivation behind learning,

which can anything be such as knowing something new,

develop projects using Python, switch to Python, etc. B

elow are the general areas where Python is widely used.

Pick any of them.

Data Analysis and Processing

Artificial Intelligence

Games

Hardware/Sensor/Robots

Desktop Applications

2. Learn the Basic Syntax

It is the most essential and basic step to learn the

syntax of the Python programming language. We have to

learn the basic syntax before dive deeper into learning

it. Python is easy to learn and has a simple syntax.

It doesn't use semicolon and brackets. Its syntax is

like the English language.

So it will take minimum amount of time to learning its

syntax. Once we get its syntax properly, further

learning will be easier and quicker getting

to work on projects.

3. Write Code by Own

Writing the code is the most effective and robust way

to learn Python. First, try to write code on paper

and run in mind (Dry Run) then move to the system.

Writing code on paper will help us get familiar

quickly with the syntax and the concept store in

the deep memory. While writing the code, try to use

proper functions and suitable variables names.

There are many editors available for Python programming

which highlights the syntax related issue automatically.

So we don't need to pay lot of attention

of these mistakes.

4. Keep Practicing

The next important step is to do the practice. It needs

to implementing the Python concepts through the code.

We should be consistence to our daily coding practice.

Consistency is the key of success in any aspect of life

not only in programming. Writing code daily will help

to develop muscle memory.

We can do the problem exercise of related concepts or

solve at least 2 or 3 problems of Python. It may seem

hard but muscle memory plays large part in programing.

It will take us ahead from those who believe only the

reading concept of Python is sufficient.

5. Make Notes as Needed

Creating notes by own is an excellent method to learn

the concepts and syntax of Python. It will establish

stability and focus that helps you become a Python

developer. Make brief and concise notes with relevant

information and include appropriate examples of the

subject concerned.

6. Discuss Concepts with Other

Coding seems to be solitary activity, but we can enhance

our skills by interacting with the others. We should

discuss our doubts to the expert or friends who are

learning Python. This habit will help to get additional

information, tips and tricks, and solution of coding

problems. One of the best advantages of Python,

it has a great community. Therefore, we can also learn

from passionate Python enthusiasts.

7. Do small Projects

After understanding Python's basic concept, a beginner

should try to work on small projects. It will help to

understand Python more deeply and become more component

in it. Theoretical knowledge is not enough to get

command over the Python language. These projects can

be anything as long as they teach you something. You

can start with the small projects such as calculator

app, a tic-toc-toe game, an alarm clock app, a to-do

list, student or customer management system, etc.

Once you get handy with a small project, you can

easily shift toward your interesting domain (Machine

Learning, Web Development, etc.).

8. Teach Others

There is a famous saying that "If you want to learn

something then you should teach other". It is also

true in case of learning Python. Share your

information to other students via creating blog

posts, recording videos or taking classes in local

training center. It will help us to enhance the

understanding of Python and explore the unseen

loopholes in your knowledge. If you don't want

to do all these, join the online forum and post

your answers on Python related questions.

9. Explore Libraries and Frameworks

Python consists of vast libraries and various

frameworks. After getting familiar with Python's

basic concepts, the next step is to explore the

Python libraries. Libraries are essential to work

with the domain specific projects. In the following

section, we describe the brief introduction of the

main libraries.

TensorFlow - It is an artificial intelligence library

which allows us to create large scale AI based projects.

Django - It is an open source framework that allows us

to develop web applications. It is easy, flexible,

and simple to manage. Flask - It is also an open

source web framework. It is used to develop lightweight

web applications. Pandas - It is a Python library

which is used to perform scientific computations.

Keras - It is an open source library, which is used

to work around the neural network.

10. Contribute to Open Source

As we know, Python is an open source language that means

it is freely available for everyone. We can also

contribute to Python online community to enhance our

knowledge. Contributing to open source projects is

the best way to explore own knowledge. We also

receive the feedback, comments or suggestions for work

that we submitted. The feedback will enable the best

practices for Python programming and help us to

become a good Python developer.

Types of Applications

1) Web Applications

We can use Python to develop web applications. It

provides libraries to handle internet protocols such

as HTML nd XML, JSON, Email processing, request,

beautifulSoup, Feedparser, etc. One of Python

web-framework named Django is used on Instagram.

Python provides many useful frameworks,

and these are given below:

Django and Pyramid framework(Use for heavy applications)

Flask and Bottle (Micro-framework)

Plone and Django CMS (Advance Content management)

2) Desktop GUI Applications

The GUI stands for the Graphical User Interface, which

provides a smooth interaction to any application. Python

provides a Tk GUI library to develop a user interface.

Some popular GUI libraries are given below.

Tkinter or Tk

wxWidgetM

Kivy (used for writing multitouch applications )

PyQt or Pyside

3) Console-based Application

Console-based applications run from the command-line

or shell. These applications are computer program

which are used commands to execute. This kind of

application was more popular in the old generation

of computers. Python can develop this kind of

application very effectively. It is famous for

having REPL, which means the Read-Eval-Print Loop

that makes it the most suitable language for the

command-line applications.

Python provides many free library or module which helps to build the

command-line apps. The necessary IO libraries are

used to read and write. It helps to parse argument and create console

help text out-of-the-box. There are also advance

libraries that can develop independent console apps.

4) Software Development

Python is useful for the software development process.

It works as a support language and can be used to

build control and management, testing, etc.

SCons is used to build control.

Buildbot and Apache Gumps are used for automated

continuous compilation and testing.

Round or Trac for bug tracking and project management.

5) Scientific and Numeric

This is the era of Artificial intelligence where the

machine can perform the task the same as the human.

Python language is the most suitable language for

Artificial intelligence or machine learning. It

consists of many scientific and mathematical

libraries, which makes easy to solve complex

calculations.

Implementing machine learning algorithms require

complex mathematical calculation. Python has many

libraries for scientific and numeric such as Numpy,

Pandas, Scipy, Scikit-learn, etc. If you have some

basic knowledge of Python, you need to import

libraries on the top of the code. Few popular

frameworks of machine libraries are given below.

SciPy

Scikit-learn

NumPy

Pandas

Matplotlib

seaborn

6) Business Applications

Business Applications differ from standard applications.

E-commerce and ERP are an example of a business

application. This kind of application requires

extensively, scalability and readability, and Python

provides all these features.

7) Audio or Video-based Applications

Python is flexible to perform multiple tasks and can be

used to create multimedia applications. Some multimedia

applications which are made by using Python are

TimPlayer, cplay, etc. The few multimedia libraries

are given below.

Gstreamer

Pyglet

QT Phonon

8) 3D CAD Applications

The CAD (Computer-aided design) is used to design

engineering related architecture. It is used to

develop the 3D representation of a part of a system.

Python can create a 3D CAD application by using the

following functionalities.

Fandango (Popular )

CAMVOX

HeeksCNC

AnyCAD

RCAM

9) Enterprise Applications

Python can be used to create applications that can be

used within an Enterprise or an Organization. Some

real-time applications are OpenERP, Tryton, Picalo, etc.

10) Image Processing Application

Python contains many libraries that are used to work

with the image. The image can be manipulated according

to our requirements. Some libraries of image

processing are given below.

OpenCV

Pillow

SimpleITK

----------------------------------------------------------------

First Python Program

In this Section, we will discuss the basic syntax of

Python, we will run a

simple program to print Hello

World on the console.

Python provides us the two ways to run a program:

Using Interactive interpreter prompt

Using a script file

Interactive interpreter prompt

Python provides us the feature to execute the Python statement one by one at

the interactive prompt. It is preferable in the case where we are concerned

about the output of each line of our Python program.

Using a script file (Script Mode Programming)

The interpreter prompt is best to run the single-line statements of the code.

However, we cannot write the code

every-time on the terminal. It is not suitable to write multiple lines of code.

Using the script mode, we can write multiple lines code into a file which can

be executed later. For this purpose,

we need to open an editor like notepad, create a file named and save it with

.py extension, which stands for "Python".

Now, we will implement the above example using the script mode.

print("hello world") #here, we have used print() function to print the message

on the console.

----------------------------

Multi-line Statements

Multi-line statements are written into the notepad like an editor and saved it

with .py extension. In the following

example, we have defined the execution of the multiple code lines using the

Python script.

---------------------------

print("Hai to python")

print("OOPS")

print("Data science")

output:

Hai to python

OOPS

Data science

------------------------------------------------

**Python Variables**

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

It is recommended to use lowercase letters for the variable name.

Rahul and rahul both are two different variables.

Identifier Naming Variables are the example of

identifiers. An Identifier is used to identify

the literals used in the program.

The rules to name an identifier are given below.

-------------------

The first character of the variable must be an alphabet

or underscore ( \_ ).

All the characters except the first character may be an

alphabet of lower-case(a-z), upper-case (A-Z),

underscore, or digit (0-9).

Identifier name must not contain any white-space,

or special character (!, @, #, %, ^, &, \*).

Identifier name must not be similar to any keyword

defined in the language.

Identifier names are case sensitive; for example,

myname, and MyName is not the same.

Examples of valid identifiers(variable): a123, \_n, n\_9, etc.

Examples of invalid identifiers: 1a, n%4, n 9, etc.

---------------------------------------------

Declaring Variable and Assigning Values

Python does not bind us to declare a variable before

using it in the application.

It allows us to create a variable

at the required time.

We don't need to declare explicitly variable in Python.

When we assign any value to the variable, that

variable is declared automatically.

The equal (=) operator is used to assign value to a

variable.

In Python, variables are a symbolic name that is a

reference or pointer to an object.

The variables are used to denote

objects by that name.

Let's understand the following example

a = 50

b =100

shabeeb = 90

------------------------------------------

**Variable Names**

We have already discussed how to declare the valid

variable. Variable names can be any length can have

uppercase, lowercase (A to Z, a to z), the digit (0-9),

and underscore character(\_). Consider the following

example of valid variables names.

-------------------

name = "Devansh"

age = 20

marks = 80.50

print(name)

print(age)

print(marks)

output:

Devansh

20

80.5

-------------------------------------------------

Consider the following valid variables name.

name = "Hai"

Name = "Boy"

naMe = "C"

NAME = "D"

n\_a\_m\_e = "E"

\_name = "F"

name\_ = "G"

\_name\_ = "H"

na56me = "I"

print(name,Name,naMe,NAME,n\_a\_m\_e, NAME, n\_a\_m\_e, \_name, name\_,\_name, na56me)

----------------------------------------

Camel Case :- In the camel case, each word or abbreviation in the middle of

begins with a capital letter.

There is no intervention of whitespace.

For example - nameOfStudent,

valueOfVaraible, etc.

Pascal Case :- It is the same as the Camel Case, but

here the first word is also capital. For example -

NameOfStudent, etc.

Snake Case :- In the snake case, Words are separated

by the underscore.

For example - name\_of\_student, etc.

**-----------------------------**

**Multiple Assignment**

Python allows us to assign a value to multiple variables

in a single statement,

which is also known as multiple assignments.

We can apply multiple assignments in two ways, either by assigning a single value

to multiple variables or assigning

multiple values to multiple variables.

Assigning single value to multiple variables

-------------------

x=y=z=50

print(x)

print(y)

print(z)

output:

50

50

50

---------------

Assigning multiple values to multiple variables:

-----------------------

a,b,c=5,10,15

print(a)

print (b)

print (c)

output:

5

10

15

-----------------------------------

**Delete a variable**

We can delete the variable using the del keyword. The syntax is given below.

Syntax -

del <variable\_name>

----------------

# Assigning a value to x

x = 6

print(x)

# deleting a variable.

del x

print(x) # error

---------------------------------

Print Single and Multiple Variables in Python

We can print multiple variables within the single print statement.

------------------

# printing single value

a = 5

print(a)

output:

5

---------------------------

**#Printing Multiple Variables**

a = 5

b = 6

# printing multiple variables

print(a,b)

# separate the variables by the comma

print(1, 22, 3, 4, 5, 6, 7, 8)

output:

5 6

1 22 3 4 5 6 7 8

---------------------------------------

**Python Data Types**

Variables can hold values, and every value has a

data-type. Python is a dynamically

typed language; hence we do not need to define the

type of the variable while declaring it. The

interpreter implicitly binds the value with its type.

a = 5

The variable a holds integer value five and we did not

define its type. Python interpreter will automatically

interpret variables a as an integer type. Python

enables us to check the type of the variable used in

the program. Python provides us the type() function,

which returns the type of the variable passed.

------------------

a=10

b="Hi Python"

c = 10.5

print(type(a))

print(type(b))

print(type(c))

output:

<class 'int'>

<class 'str'>

<class 'float'>

-------------------------------------

**Standard data types**

A variable can hold different types of values. For example, a person's

name must be stored as a string whereas its id must be stored as

an integer.

Python provides various standard data types that define the storage method on

each of them. The data types defined in Python are given below.

Numbers

Sequence Type

Boolean

Set

Dictionary

-----------------------

Numbers

Number stores numeric values. The integer, float, and complex values belong

to a Python Numbers data-type. Python provides the type() function to

know the data-type of the variable.

a = 5

print("The type of a", type(a))

b = 40.5

print("The type of b", type(b))

c = 1+3j

print("The type of c", type(c))

output:

The type of a <class 'int'>

The type of b <class 'float'>

The type of c <class 'complex'>

-------------------------

Int - Integer value can be any length such as integers 10, 2, 29, -20,

-150 etc. Python has no restriction on the length of an integer.

Its value belongs to int Float - Float is used to store floating-point

numbers like 1.9, 9.902, 15.2, etc.

It is accurate upto 15 decimal points.

complex - A complex number contains an ordered pair, i.e., x + iy

where x and y denote the real and imaginary parts, respectively.

The complex numbers

like 2.14j, 2.0 + 2.3j, etc.

--------------------------------------

Python Keywords

Python Keywords are special reserved words that convey a special meaning

to the compiler/interpreter. Each keyword

has a special meaning and a specific operation. These keywords can't be

used as a variable. Following is the List

of Python Keywords.

True False None and as

asset def class continue break

else finally elif del except

global for if from import

raise try or return pass

nonlocal in not is lambda

-------------------------------------

**Python Literals**

Python Literals can be defined as data that is given in a

variable or constant.

String literals:

String literals can be formed by enclosing a text in the

quotes. We can use both single as well as double quotes

to create a string. Types of Strings:

There are two types of Strings supported in Python:

Single-line String- Strings that are terminated within a single-line

are known as Single line Strings.

Example:

-----------------

tx='hello to python'

print(tx)

output:

hello to python

--------------------

Multi-line String - A piece of text that is written in multiple lines

is known as multiple lines string.

There are two ways to create multiline strings:

1) Adding black slash at the end of each line.

------------

a=" India \

is my \

country"

print(a)

output:

India is my country

-------------------------

2) Using triple quotation marks:-

a=""" India

is my

country"""

print(a)

output:

India

is my

country

------------------

b=''' India

is my

country'''

print(b)

output:

India

is my

country

-------------------------

Numeric literals:

Int(signed integers)

Numbers( can be both positive and negative) with no

fractional part.eg: 100

-----------

Long(long integers)

Integers of unlimited size followed by lowercase or

uppercase L eg: 87032845L

float(floating point)

Real numbers with both integer and fractional part

eg: -26.2

--------------------

#numericals

x = 0b10100 # Binary Literals

y = 129 # Decimal Literal

z = 0o215 # Octal Literal

u = 0x12d # Hexadecimal Literal

print(x)

print(y)

print(z)

print(u)

output:

20

129

141

301

------------------

# Float Literal

float\_1 = 100.5

float\_2 = 1.5e2

print(float\_1)

print(float\_2)

output:

100.5

150.0

-------------------

# Complex Literal

a = 5 + 3.14j

print(a, a.imag, a.real)

output:

(5+3.14j) 3.14 5.0

------------------------

II. Boolean literals:

A Boolean literal can have any of the two values: True or

False.

-------------

a=True

b=False

print(a)

print(b)

print(type(a))

print(type(b))

output:

True

False

<class 'bool'>

<class 'bool'>

---------------------

**Python Operators**

The operator can be defined as a symbol which is

responsible for a particular operation between two

operands. Operators are the pillars of a program

on which the logic is built in a specific programming

language. Python provides a variety of operators,

which are described

as follows.

Arithmetic operators

Comparison operators

Assignment Operators

Logical Operators

Membership Operators

Identity Operators

----------------------

Python Arithmetic Operators

+ (Addition) It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30

- (Subtraction) It is used to subtract the second operand from the first operand. If the first operand

is less than the second operand, the value results negative. For example, if a = 20, b = 10 => a - b = 10

/ (divide) It returns the quotient after dividing the first operand by the second operand.

For example, if a = 20, b = 10 => a/b = 2.0

\* (Multiplication) It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a \* b = 200

% (reminder) It returns the reminder after dividing the first operand by the second operand.

For example, if a = 10, b = 4 => a%b = 2

\*\* (Exponent)5\*\*2 => 25 It is an exponent operator represented as it calculates the first operand power to the second operand.

// (Floor division) 5//2 =>2 It gives the floor value of the quotient produced by dividing the two operands.

--------------------

x = 5

y = 3

print(x + y)

output:

8

----------------------

x = 5

y = 3

print(x - y)

------------------

x = 5

y = 3

print(x \* y)

--------------

x = 12

y = 3

print(x / y)

-----------------

x = 5

y = 2

print(x % y)

output:

1

----------

x = 2

y = 5

print(x \*\* y) #same as 2\*2\*2\*2\*2

----------------

x = 15

y = 2

print(x // y) #7

#the floor division // rounds the result down to the nearest whole numbe

--------------------------

Comparison operator

Comparison operators are used to comparing the value of the two operands

and returns Boolean true or false accordingly.

The comparison operators are described in the following table.

== If the value of two operands is equal, then the condition becomes true.

!= If the value of two operands is not equal, then the condition becomes true.

<= If the first operand is less than or equal to the second operand, then the condition becomes true.

>= If the first operand is greater than or equal to the second operand, then the condition becomes true.

> If the first operand is greater than the second operand, then the condition becomes true.

< If the first operand is less than the second operand, then the condition becomes true.

----------------------

x = 5

y = 3

print(x == y)

# returns False because 5 is not equal to 3

----------------------

x = 5

y = 3

print(x != y)

# returns True because 5 is not equal to 3

---------------

x = 5

y = 3

print(x > y)

# returns True because 5 is greater than 3

-----------------

x = 5

y = 3

print(x < y)

# returns False because 5 is not less than 3

------------------------

x = 5

y = 3

print(x >= y)

# returns True because five is greater, or equal, to 3

---------------------------

x = 5

y = 3

print(x <= y)

# returns False because 5 is neither less than or equal to 3

----------------------

Assignment Operators

The assignment operators are used to assign the value of the right expression to the left operand.

The assignment operators are described in the following table.

= It assigns the value of the right expression to the left operand.

+= It increases the value of the left operand by the value of the right operand and assigns the modified value

back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.

-= It decreases the value of the left operand by the value of the right operand and assigns the modified value

back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.

\*= It multiplies the value of the left operand by the value of the right operand and assigns the modified value

back to then the left operand. For example, if a = 10, b = 20 => a\* = b will be equal to a = a\* b and therefore, a = 200.

%= It divides the value of the left operand by the value of the right operand and assigns the reminder

back to the left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.

\*\*= a\*\*=b will be equal to a=a\*\*b, for example, if a = 4, b =2, a\*\*=b will assign 4\*\*2 = 16 to a.

//= A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.

----------------------

x = 5

print(x)

output:

5

---------------

x = 5

x += 3 #same as x=x+3

print(x)

output:

8

----------------

x = 5

x -= 3 #same as x=x-3

print(x)

output:

2

----------------

x = 5

x \*= 3 #same as x=x\*3

print(x)

output:

15

----------------

x = 5

x /= 3 vvvv#same as x=x/3

print(x)

---------------------

Python Logical Operators

Logical operators are used to combine conditional

statements:

Logical operators,and, or not.

-----------------

x = 5

print(x > 3 and x < 10)

# returns True because 5 is greater than 3 AND 5 is

less than 10

---------------

True and True => True

True and False => False

False and True => False

False and Fals => False

---------------------------

or Returns True if one of the statements is true x < 5 or x < 4

----------------

x = 5

print(x > 3 or x < 4)

# returns True because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)

-------

'or' working

True or True => True

True or False => True

False or True => True

False or False => False

-----------------------------

not Reverse the result, returns False if the result is true not(x < 5 and x < 10)

x = 5

print(not(x > 3 and x < 10))

# returns False because not is used to reverse the result

------------

'not' meaning

not(True) =>False

not(False)=> True

----------------------

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object,

with the same memory location:

is Returns True if both variables are the same object x is y

-----------

st1="hello"

st2="hello"

print(st1 is st2)

output:

True

---------

st1="hello"

st2="Hello"

print(st1 is st2)

output:

False

-------------

is not Returns True if both variables are not the same object x is not y

------------------

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

print("is" in "india is my country")

output:

True

**------------------**

**Python Comments**

Python Comment is an essential tool for the programmers. Comments are

generally used to explain the code.

We can easily understand the code if it has a proper explanation.

A good programmer must use the comments

because in the future anyone wants to modify the code as well as implement

the new module; then, it can be done easily.

In the other programming language such as C++, It provides the //

for single-lined comment and /\*.... \*/ for multiple-lined comment,

but Python provides the single-lined Python comment. To apply the

comment in the code we use the hash(#) at the beginning of the

statement or code.

-----------------

# This is the print statement

#print("hai"")

print("Hello Python")

output:

Hello Python

-------------------

Multiline Python Comment

We must use the hash(#) at the beginning of every line of code to apply

the multiline Python comment.

------------------

# First line of the comment

# Second line of the comment

# Third line of the comment

#print("hello")

print("okay")

output:

okay

----------------

# Variable a holds value 5

# Variable b holds value 10

# Variable c holds sum of a and b

# print("hey")

a = 5

b = 10

c = a+b

print("The sum is:", c)

output:

15

---------------------

We can also use the triple quotes (''') or (""") for multiline comment.

The triple quotes are also used to string formatting.

Eg: Multiline comments

------------------

"""Variable a holds value 5

Variable b holds value 10

Variable c holds sum of a and b

Print the result"""

'''Variable a holds value 5

Variable b holds value 10

Variable c holds sum of a and b

Print the result'''

a = 5

b = 10

c = a+b

print("The sum is:", c)

output:

15

#----------------------------------------

**Taking input in Python**

Developers often have a need to interact with users, either to get data or

to provide some sort of result.

Syntax:

input( prompt message )

-------------

# Python program showing

# a use of input()

abc = input("Enter your value: ")

print(abc)

output:

Enter your value: kochi

kochi

-----------------------------

**Python indentation**

Python indentation uses to define the block of the

code. The other programming languages such as C,

C++, and Java use curly braces {}, whereas Python

uses an indentation. Whitespaces are used as

indentation in Python.

Indentation uses at the beginning of the code and

ends with the unintended line. That same line

indentation defines the block of the code

(body of a function, loop, etc.)

Generally, four whitespaces are used as the

indentation. The amount of indentation depends on

user, but it must be consistent throughout that

block.

Block1:

stat1

stat2

...

Block11:

Stat1

Stat2

...

Block2:

Stat1

Stat2

...

--------------------------------

**Python If-else statements**

Decision making is the most important aspect of almost all the programming

languages. As the name implies, decision making allows us to run a

particular block of code for a particular decision. Here, the decisions

are made on the validity of the particular conditions. Condition

checking is the backbone of decision making.

-----------------

If Statement The if statement is used to test a specific condition. If the

condition is true, a block of code (if-block) will be executed.

If - else Statement The if-else statement is similar to if statement except

the fact that, it also provides the block of the code for the false case of

the condition to be checked. If the condition provided in the if statement

is false, then the else statement will be executed.

Nested if Statement Nested if statements enable us to use if ? else statement

inside an outer if statement.

----------------------

The simple if statement:

The syntax of the if-statement is given below.

if expression:

statement

The if statement is used to test a particular condition and if the condition is

true, it executes a block of code known as if-block. The condition of if statement

can be any valid logical expression which can be either evaluated to true or false.

---------------------

#Simple if Example:

#to check given no. is odd or even

num = int(input("enter the number:"))

if num%2 == 0:

print("Number is even")

if num%2 != 0:

print("Number is odd")

output:

enter the number:12

Number is even

--------------------

#Program to print the largest of the three numbers.

a = int(input("Enter a: "))

b = int(input("Enter b: "))

c = int(input("Enter c: "))

if a>b and a>c:

print("a is largest")

if b>a and b>c:

print("b is largest")

if c>a and c>b:

print("c is largest")

output:

Enter a: 10

Enter b: 20

Enter c: 30

c is largest

--------------------

The if-else statement

The if-else statement provides an else block combined with the if statement

which is executed in the false case of the condition.

If the condition is true, then the if-block is

executed. Otherwise, the else-block is executed.

The syntax of the if-else statement is given below.

if condition:

#block of statements

else:

#another block of statements (else-block)

----------------

#Program to check whether a person is eligible to

vote or not.

age = int(input("Enter your age:"))

if age>=18:

print("You are eligible to vote !!")

else:

print("Sorry! you have to wait !!")

output:

Enter your age:17

Sorry! you have to wait !!

---------------------------

Program to check whether a number is even or not.

num = int(input("enter the number:"))

if num%2 == 0:

print("Number is even...")

else:

print("Number is odd...")

output:

enter the number:30

Number is even...

--------------------------------

The elif statement

The elif statement enables us to check multiple conditions and execute the

specific block of statements depending upon the true condition among them.

We can have any number of elif statements in our program

depending upon our need. However, using elif is optional.

#syntax

if expression 1:

# block of statements

elif expression 2:

# block of statements

elif expression 3:

# block of statements

...

...

...

else:

# block of statements

-----------------------

# example elif

marks = int(input("Enter the marks? "))

if marks > 85 and marks <= 100:

print("Congrats ! you scored grade A ...")

elif marks > 60 and marks <= 85:

print("You scored grade B + ...")

elif marks > 40 and marks <= 60:

print("You scored grade B ...")

elif (marks > 30 and marks <= 40):

print("You scored grade C ...")

else:

print("Sorry you are fail ?")

-----------------------

# example elif

number = int(input("Enter the number?"))

if number==10:

print("number is equals to 10")

elif number==50:

print("number is equal to 50")

elif number==100:

print("number is equal to 100")

else:

print("number is not equal to 10, 50 or 100")

---------------------------------

Nested if Statement

if statement can also be checked inside other if statement. This conditional

statement is called a nested if statement. This means that inner if condition

will be checked only if outer if condition is true and by this,

we can see multiple conditions to be satisfied.

----------------------

Nested if Syntax:

if (condition1):

# Executes when condition1 is true

if (condition2):

# Executes when condition2 is true

# if Block is end here

# if Block is end here

-------------------

#largest of 3-numbers using nested if

a = int(input("Enter the a:"))

b = int(input("Enter the b:"))

c = int(input("Enter the c:"))

if a > b:

if a>c:

print("a is larger")

else:

print("c is larger")

else:

if b>c:

print("b is larger")

else:

print("c is larger")

-----------------------------

**Python Loops:**

The flow of the programs written in any programming language is sequential

by default. Sometimes we may need to alter the flow of the program. The execution

of a specific code may need to be repeated several numbers of times. For this

purpose, The programming languages provide various types of loops which are

capable of repeating some specific code several numbers of times.

Why we use loops in python?

The looping simplifies the complex problems into the easy ones. It enables us

to alter the flow of the program so that instead of writing the same code again

and again, we can repeat the same code for a finite number of times. For example,

if we need to print the first 10 natural numbers then, instead of using the print

statement 10 times, we can print inside a loop which runs up to 10 iterations.

------------------------------

Advantages of loops:

There are the following advantages of loops in Python.

1. It provides code re-usability.

2. Using loops, we do not need to write the same code again and again.

3. Using loops, we can traverse over the elements of data structures (array or

linked lists).

--------------------------------

Python 'for' loops:

The for loop in Python is used to iterate the statements or a part of the program

several times.

It is frequently used to traverse the data structures like list, tuple, or

dictionary.

The syntax of for loop in python is given below.

for iterating\_var in sequence:

statement(s)

---------------

str = "Python"

for i in str:

print(i)

output:

P

y

t

h

o

n

---------------------

For loop Using range() function

The range() function

The range() function is used to generate the sequence of the numbers. If we

pass the range(10), it will generate the numbers from 0 to 9. The syntax of

the range() function is given below.

Syntax of range():

range(start,stop,step size)

The start represents the beginning of the iteration.

The stop represents that the loop will iterate till stop-1.

The step size is used to skip the specific numbers from the iteration.

It is optional to use.

By default, the step size is 1. It is optional.

--------------------------------

#Program to print numbers in sequence.

for i in range(10):

print(i, end = ' ')

output:

0 1 2 3 4 5 6 7 8 9

------------------

#Program to print table of given number.

n = int(input("Enter the number:"))

for i in range(1,11):

c = n\*i

print(n,"x",i,"=",c)

output:

Enter the number:5

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

---------------------------

#Program to print even number using step size in range().

n = int(input("Enter the limit number:"))

for i in range(2,n,2):

print(i)

output:

Enter the limit number:20

2

4

6

8

10

12

14

16

18

-----------------------

Nested for loop in python:

Python allows us to nest any number of for loops inside a for loop. The inner

loop is executed n number of times for every iteration of the outer loop.

Syntax:

for iterating\_var1 in sequence: #outer loop

for iterating\_var2 in sequence: #inner loop

#block of stat

---------------

#pattern printing

# User input for number of rows

"""

\*

\*\*

\*\*\*

\*\*\*\*

"""

---------------------

rows = int(input("Enter the rows:"))

# Outer loop will print number of rows

for i in range(1,rows+1):

# Inner loop will print number of Astrisk

for j in range(i):

print("\*",end = '')

print()

output:

Enter the rows:7

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*

--------------

1

22

333

4444

55555

----------------

1

12

123

1234

-----------------------

Using else statement with for loop:

Unlike other languages like C, C++, or Java, Python allows us to use the

else statement with the for loop which can be executed only when all the

iterations are exhausted. Here, we must notice that if the loop contains

any of the break statement then the else statement will not be executed.

-------------------------

for i in range(0,5):

print(i)

break

else:

print("for loop is exhausted")

print("The loop is broken due to break statement...came out of the loop")

output:

0

The loop is broken due to break statement...came out of the loop

--------------------------------

Python While loop:

The Python while loop allows a part of the code to be executed until the given

condition returns false. It is also known as a pre-tested loop.

It can be viewed as a repeating if statement. When we don't know the number of

iterations then the while loop is most effective to use.

The syntax of while loop:

while expression:

statements

-----------------

#print numbers from 0 to 8

i=0

while i<8:

print(i)

i=i+1

output:

0

1

2

3

4

5

6

7

---------------

Using else with while loop:

Python allows us to use the else statement with the while loop also. The else

block is executed when the condition given in the while statement becomes false.

Like for loop, if the while loop is broken using break statement, then the else

block will not be executed, and the statement present after else block will be

executed.

The else statement is optional to use with the while loop.

---------------------

#else with while

i=0

while i<8:

print(i)

i=i+1

else:

print(" no breaks")

print("outside loop")

output:

0

1

2

3

4

5

6

7

no breaks

outside loop

---------------

#else with while

i=0

while i<25:

if i==5:

break

print(i)

i=i+1

else:

print(" no breaks")

print("outside loop")

output:

0

1

2

3

4

outside loop

----------------

**Loop Control Statements:**

We can change the normal sequence of while loop's execution using the loop

control statement. When the while loop's execution is completed, all automatic

objects defined in that scope a re demolished. Python offers the following

control statement to use within the while loop.

Break Statement - When the break statement is encountered, it brings control

out of the loop.

# The control transfer is transfered

# when break statement soon it sees

------------------------------

#use of continue statement

# prints all numbers except 10

i = 0

while i < 30:

if i == 10:

i =i+ 1

continue

print('Current number :', i)

i += 1

-------------------------

# execution breaks when reading reaches at 15

i = 0

while i < 30:

if i == 15:

break

print('Current number :', i)

i += 1

print("outside while")

-----------------------

**Python String:**

Till now, we have discussed numbers as the standard data-types in Python.

In this section , we will discuss the most popular data type in Python, i.e.,

string. Python string is the collection of the characters surrounded by

single quotes, double quotes, or triple quotes.

Consider the following example in Python to create a string.

Syntax:

str="Hi python"

Here, if we check the type of the variablestrusing a Python script

print(type(str)) then it will print as <class 'str'>.

-------------

Creating String in Python:

We can create a string by enclosing the characters in single-quotes or

double- quotes. Python also provides triple-quotes to represent the string,

-------------------

# Using single quotes

str1 = 'Hello Python'

print(str1)

output:

Hello Python

-----------------

# Using double quotes

str2 = "Hello Python"

print(str2)

output:

Hello Python

--------------

# Using triple quotes

str3 = '''Triple quotes are generally used for

represent the multiline or

docstring'''

print(str3)

output:

Triple quotes are generally used for

represent the multiline or

docstring

-----------------

# Using triple quotes

str4 = """Triple quotes are generally used for

represent the multiline or

docstring"""

print(str4)

output:

Triple quotes are generally used for

represent the multiline or

docstring

--------------------------

Strings indexing and splitting

Like other languages, the indexing of the Python strings starts from 0.

-----------------

str = "HELLO"

print(str[0]) # returns H

print(str[1]) # returns E

print(str[2]) # returns L

print(str[3]) # returns L

print(str[4]) # returns O

# It returns the IndexError because 6th index doesn't exist

print(str[6]) #error

-------------

slice operator [] is used to access the individual characters of the string.

However, we can use the : (colon) operator in Python

to access the substring from the given string.

# Given String

str = "INTERNATIONAL"

# Start Oth index to end

print(str[0:])

# Starts 1th index to 4th index

print(str[1:5])

# Starts 2nd index to 3rd index

print(str[2:4])

# Starts 0th to 2nd index

print(str[:3])

#Starts 4th to 6th index

print(str[4:7])

-------------------

We can do the negative slicing in the string; it starts from the rightmost

character, which is indicated as -1.

The second rightmost index indicates -2, and so on.

# Given String

str = "INTERNATIONAL"

print(str[-1]) # L

print(str[-3]) # N

print(str[-2:]) #AL

print(str[-4:-1]) #ONA

print(str[-7:-2]) #ATION

# Reversing the given string

print(str[::-1]) #reverse

print(str[-12]) #N

------------------

Reassigning Strings

Updating the content of the strings is as easy as assigning it to

a new string. The string object doesn't support item assignment

i.e., A string can only be replaced with new string since its content

cannot be partially replaced.

-------------

str = "HELLO"

print(str)

str = "hello"

print(str)

output:

HELLO

hello

---------

but,

-------

str = "HELLO"

str[0] = "h" #error

print(str)

---------------

Deleting the String:

As we know that strings are immutable. We cannot delete or remove the characters

from the string. But we can delete the entire string using the del keyword.

--------------

str = "python"

del str[1] #raise error

---------

Now we are deleting entire string.

str1 = "python"

del str1

print(str1) #error

-------------------

String Operators:

+ ,It is known as concatenation operator used to join the strings

given either side of the operator.

\* ,It is known as repetition operator. It concatenates the

multiple copies of the same string.

[] ,It is known as slice operator. It is used to access the

sub-strings of a particular string.

[:] ,It is known as range slice operator. It is used to access

the characters from the specified range.

in ,It is known as membership operator. It returns if a particular sub-string is

present in the specified string. not in ,It is also a membership

operator and does the exact reverse of in. It returns true

if a particular substring is not present in the specified string.

r/R ,It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of

escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.

% ,It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f

to map their values in python. We will discuss how formatting is done in python.

------------------------------

str = "Hello"

str1 = " world"

print(str\*3) #prints HelloHelloHello

print(str+str1) #prints Hello world

print(str[4]) #prints o

print(str[2:4]) #prints ll

print('w' in str) #prints false as w is not present in str

print('wo' not in str1) # prints false as wo is present in str1.

print(r'C://python37') # prints C://python37 as it is written

print("The string str : %s and str1:%s" % (str,str1)) # prints The

string str : Hello

---------------------------------------------------------------

**Python String Formatting:**

Escape Sequence

Let's suppose we need to write the text as - They said, "Hello what's going on?"-

the given statement can be written in single quotes or double quotes but it will

raise the SyntaxError as it contains both single and double-quotes.

str = "They said, "Hello what's going on?"" #error

print(str)

-------------------------------------------------------

The backslash(\) symbol denotes the escape sequence. The backslash can be

followed by a special character and it interpreted differently. The single

quotes inside the string must be escaped. We can apply the same as in the double quotes.

The list of an escape sequence:

#----------------------------------

\ It ignores the new line

print("Python1 \

Python2 \

Python3")

output:

Python1 Python2 Python3

#-----------------------

\\ Backslash

print("\\")

output:

\

#---------------------

\' Single Quotes

print('\'')

output:

'

#------------------------

\" Double Quotes

print("\"")

output:

"

#----------------------------

\b ASCII Backspace(BS)

print("Hello \b World")

#-------------------------

\n new line

print("Hello \n World!")

output:

Hello

World!

#--------------------------

\t ASCII Horizontal Tab

print("Hello \t World!")

output:

Hello World!

#-----------------------------------

We can ignore the escape sequence from the given string by using the raw string. We can do this

by writing r or R in front of the string.

print(r"C:\\Users\\DEVANSH SHARMA\\Python32")

output:

C:\\Users\\DEVANSH SHARMA\\Python32

#-------------------------------------

Python String Formatting Using % Operator:

Python allows us to use the format specifiers used in C's printf statement. The

format specifiers in Python are treated in the same way as they are treated in C.

However, Python provides an additional operator %, which is used as an

interface between the format specifiers and their values. In other words,

we can say that it binds the format specifiers to the values.

-------------------------------

var1 = 10

var2 = 1.290

var3 = "Devansh"

print("Integer: %d\nFloat:%f\n arString:%s" % (var1,var2,var3))

output:

Integer: 10

Float:1.290000

String:Devansh

----------------------------------

Python String functions:

Python String capitalize() Method

Python capitalize() method converts first character of the string into uppercase

without altering the whole string.

It changes the first character only and skips rest of the string unchanged.

Signature:

capitalize()

Parameters:

No parameter is required.

Return Type

It returns a modified string.

# Python capitalize() function example

------------------

# Variable declaration

str = "python data"

# Calling function

str2 = str.capitalize()

# Displaying result

print("Old value:", str)

print("New value:", str2)

output:

Old value: python data

New value: Python data

----------------------------------------

**Python String Count() Method**

It returns the number of occurences of substring in the specified range. It takes

three parameters, first is a substring, second a start index and third is last

index of the range. Start and end both are optional whereas substring is required.

Signature

count(sub, start, end)

Parameters

sub (required)

start (optional)

end (optional)

Return Type

It returns number of occurrences of substring in the range.

----------------

# Python count() function example

# Variable declaration

str = "Hello python"

str2 = str.count('o')

# Displaying result

print("occurences:", str2)

output:

2

----------------------------------

Python String endswith() Method

Python endswith() method returns true of the string ends with the specified

substring, otherwise returns false.

Signature

endswith(suffix, start, end)

Parameters

suffix : a substring

start : start index of a range

end : last index of the range

Start and end both parameters are optional.

Return Type

It returns a boolean value either True or False.

------------------

# Python endswith() function example

# Variable declaration

str = "Hello this is python"

isends = str.endswith("on")

# Displaying result

print(isends)

output:

True

----------------------

Python String find() Method

Python find() method finds substring in the whole string and returns index of

the first match. It returns -1 if substring does not match.

Signature

find(sub, start, end)

Parameters

sub : substring

start : start index a range

end : last index of the range

Return Type

If found it returns index of the substring, otherwise -1.

---------------------

# Python find() function example

# Variable declaration

str = "Weltcome to the python"

# Calling function

str2 = str.find("to")

# Displaying result

print(str2)

output:

9

--------------------------------

Python String format() Method

Python format() method is used to perform format operations on string. While

formatting string a delimiter {} (braces) is used to replace it with the value.

This delimeter either can contain index or positional argument.

Signature

format(\*args, \*\*kwargs)

Parameters

\*args : substring

\*\*kwargs : start index a range

Return Type

It returns a formatted string.

-----------------

# Python format() function example

# Variable declaration

str = "Java"

str2 = "C#"

# Calling function

str3 = "{} and {} both are programming languages".format(str,str2)

# Displaying result

print(str3)

output:

Java and C# both are programming languages

--------------------------

Python String index() Method

Python index() method is same as the find() method

except it returns error on failure.

This method returns index of

first occurred substring and an error if there is no

match found.

Signature

index(sub, start, end)

Parameters

sub : substring

start : start index a range

end : last index of the range

Return Type

If found it returns an index of the substring, otherwise an error ValueError.

-------------------

# Python index() function example

# Variable declaration

str = "Welcome to the python"

# Calling function

str2 = str.index("py")

# Displaying result

print(str2)

output:

15

----------------------------------------

Python String isalnum() Method

Python isalnum() method checks whether the all characters of the string is

alphanumeric or not. A character which is either a letter or a number is known

as alphanumeric. It does not allow special chars even spaces.

Signature

isalnum()

Parameters

No parameter is required.

Return

It returns either True or False.

-----------

# Python isalnum() function example

# Variable declaration

str = "Welcome"

# Calling function

str2 = str.isalnum()

# Displaying result

print(str2)

output:

True

#-------------------------------

Python String isalpha() Method

Python isalpha() method returns true if all characters in the string are alphabetic. It returns False if the

characters are not alphabetic. It returns either True or False.

Signature

isalpha()

Parameters

No parameter is required.

Return

It returns either True or False

------------------------

# Python isalpha() method example

# Variable declaration

str = "python"

# Calling function

str2 = str.isalpha()

# Displaying result

print(str2)

output:

True

--------------------------------

Python String isdigit() Method

Python isdigit() method returns True if all the characters in the string are digits.

It returns False if no character

is digit in the string.

Signature

isdigit()

Parameters

No parameter is required.

Return

It returns either True or False.

-----------------

# Python isdigit() method example

# Variable declaration

str = '12345'

# Calling function

str2 = str.isdigit()

# Displaying result

print(str2)

output:

True

---------------------------

Python String islower() Method

Python string islower() method returns True if all characters in the string are

in lowercase.

It returns False if not in lowercase.

Signature

islower()

Parameters

No parameter is required.

Return

It returns either True or False.

-------------------

# Python islower() method example

# Variable declaration

str = "python"

# Calling function

str2 = str.islower()

# Displaying result

print(str2)

output:

True

#-----------------------------

Python String isupper() Method

Python isupper() method returns True if all characters in the string are in uppercase.

It returns False if characters are not in uppercase.

Signature

isupper()

Parameters

No parameter is required.

Return

It returns either True or False.

---------------

# Python isupper() method example

# Variable declaration

str = "WELCOME TO PYTHON"

# Calling function

str2 = str.isupper()

# Displaying result

print(str2)

output:

True

----------------------

Python String isspace() Method

Python isspace() method is used to check space in the string. It returna true if

there are only whitespace characters in the string. Otherwise it returns false.

Space, newline, and tabs etc are known as whitespace characters and are defined

in the Unicode character database as Other or Separator.

Signature

isspace()

Parameters

No parameter is required.

Return

It returns either True or False.

-----------------

# Python isspace() method example

# Variable declaration

str = " " # empty string

# Calling function

str2 = str.isspace()

# Displaying result

print(str2)

output:

True

#------------------------------------

Python String istitle() Method

Python istitle() method returns True if the string is a titlecased string. Otherwise

returns False.

Signature

istitle()

Parameters

No parameter is required.

Return

It returns either True or False.

-------------

# Python istitle() method example

# Variable declaration

str = "Welcome To Python"

# Calling function

str2 = str.istitle()

# Displaying result

print(str2)

output:

True

#------------------------

Python String lower() Method

Python lower() method returns a copy of the string after converting all the characters into lowercase.

Signature

lower()

Parameters

No parameter.

Return

It returns a lowercase string.

------------------

# Python lower() method example

# Variable declaration

str = "Python"

# Calling function

str = str.lower()

# Displaying result

print(str)

output:

python

-----------------------------------

Python String lstrip() Method

Python lstrip() method is used to remove all leading characters from the string. It takes a char type parameter

which is optional. If parameter is not provided, it removes all the leading spaces from the string.

Signature

lstrip([chars])

Parameters

chars (optional) : A list of chars

Return

It returns a string.

-------------------

# Python lstrip() method example

# Variable declaration

str = " python "

# Calling function

str2 = str.lstrip()

# Displaying result

print(str)

print(str2)

------------------------------

Python String replace() Method

Return a copy of the string with all occurrences of substring old replaced by new.

If the optional argument count is given, only the first count occurrences are replaced.

Signature

replace(old, new)

Parameters

old : An old string which will be replaced.

new : New string which will replace the old string.

----------------

# Python replace() method example

# Variable declaration

str = "Java is a programming language"

# Calling function

str2 = str.replace("Java","C")

# Displaying result

print("Old String: \n",str)

print("New String: \n",str2)

output:

Old String:

Java is a programming language

New String:

C is a programming language

#----------------------------------

Python String rstrip() Method

Python rstrip() method removes all the trailing characters from the string. It means

it removes all the specified characters from right side of the string. If we don't

specify the parameter, It removes all the whitespaces from the

string. This method returns a string value.

Signature

rstrip([chars])

Parameters

chars: character to be removed from the string.

Return

It returns string.

---------------------

# Python rstrip() method example

# Variable declaration

str = "Java and C# "

# Calling function

str2 = str.rstrip()

# Displaying result

print("Old string: ",str)

print("New String: ",str2)

---------------------------------

Python String split() Method

Python split() method splits the string into a comma separated list. It separates

string based on the separator delimiter. This method takes two parameters and both

are optional. It is described below.

Signature

split(sep=None)

Parameters

sep: A string parameter acts as a seperator.

------------------

# Python split() method example

# Variable declaration

str = "Java is a programming language"

# Calling function

str2 = str.split(" ")

# Displaying result

print(str)

print(str2)

output:

Java is a programming language

['Java', 'is', 'a', 'programming', 'language']

#-------------------------------------------

Python String startswith() Method

Python startswith() method returns either True or False. It returns True if the

string starts with the prefix, otherwise False. It takes two parameters start and

end. Start is a starting index from where searching starts and end index is where

searching stops.

Signature

startswith(prefix, start, end)

Parameters

prefix : A string which is to be checked.

start : Start index from where searching starts.

end : End index till there searching performs.

Both start and end are optional parameters.

Return

It returns boolean value either True or False.

--------------

# Python String startswith() method

# Declaring variable

str = "Hello Python"

# Calling function

str2 = str.startswith("Hello")

# Displaying result

print (str2)

output:

True

#-----------------------------------

strip() It is used to perform lstrip() and rstrip() on the string.

#----------------------------

Python String swapcase() Method

Python swapcase() method converts case of the string characters from uppercase to

lowercase and vice versa. It does not require any parameter and returns a string

after case conversion.

Signature

swapcase()

Parameters

No Parameter

Return

It returns a string.

----------------

# Python String swapcase() method

# Declaring variable

str = "Hello Python"

# Calling function

str2 = str.swapcase()

# Displaying result

print (str2)

output:

hELLO pYTHON

#---------------------------------

title() It is used to convert the string into the title-case i.e., The string meEruT

will be converted to Meerut.

# Declaring table and variables

str = "Hello python"

# Calling function

str2 = str.title()

# Displaying result

print(str2)

output:

Hello Python

#----------------------------------------

Python String upper() Method

Python upper() method converts all the character to uppercase and returns a uppercase

string.

Signature

upper()

Parameters

No parameters

Return

It returns a string.

--------------------------

# Python upper() method

# Declaring table and variables

str = "Hello python"

# Calling function

str2 = str.upper()

# Displaying result

print(str2)

output:

HELLO PYTHON

#--------------------------------------

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#--------------------------------------

**Python Lists**

List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data,

the other 3 are Tuple,

Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

#--------------------------

Create a List:

thislist = ["apple", "banana", "cherry"]

print(thislist)

output:

['apple', 'banana', 'cherry']

#-------------------------

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second

item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have

a defined order, and that order will

not change.

If you add new items to a list, the new items will be placed at

the end of the list.

Changeable

The list is changeable, meaning that we can change, add, and remove

items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

#----------------------------------------------------

Lists allow duplicate values:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]

print(thislist)

output:

['apple', 'banana', 'cherry', 'apple', 'cherry']

#---------------------------------------------------

List Length

To determine how many items a list has, use the len() function:

Print the number of items in the list:

thislist = ["apple", "banana", "cherry"]

print(len(thislist))

output:

3

#-----------------------------------------------------

List Items - Data Types

List items can be of any data type:

String, int and boolean data types:

list1 = ["apple", "banana", "cherry"]

list2 = [1, 5, 7, 9, 3]

list3 = [True, False, False]

print(list1)

print(list2)

print(list3)

output:

['apple', 'banana', 'cherry']

[1, 5, 7, 9, 3]

[True, False, False]

#-----------------------

A list can contain different data types:

A list with strings, integers and boolean values:

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

print(list1)

output:

['abc', 34, True, 40, 'male']

#---------------------

type()

From Python's perspective, lists are defined as objects with the

data type 'list':

<class 'list'>

data type of a list?

-------------------------

mylist = ["apple", "banana", "cherry"]

print(type(mylist))

output:

<class 'list'>

#---------------------------------------------------------------

The list() Constructor

It is also possible to use the list() constructor when creating a

new list. Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets

print(thislist)

thislist2 = list(["apple", "banana", "cherry"]) # note the double round-brackets

print(thislist2)

output:

['apple', 'banana', 'cherry']

['apple', 'banana', 'cherry']

#----------------------------------------------------

Python - Access List Items

Access Items

List items are indexed and you can access them by referring to the index number:

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]

print(thislist[1])

output:

banana

#----------------------------------

Note: The first item has index 0.

Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

Print the last item of the list:

thislist = ["apple", "banana", "cherry"]

print(thislist[-1])

output:

cherry

#----------------------------------------------------

Python - Access List Items

Access Items

List items are indexed and you can access them by referring to

the index number:

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]

print(thislist[1])

output:

banana

#----------------------------------

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

the items from "orange" (-4) to, but NOT including "mango" (-1):

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]

print(thislist[-4:-1])

output:

['orange', 'kiwi', 'melon']

#--------------------------------------------------------

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

Check if "apple" is present in the list:

thislist = ["apple", "banana", "cherry"]

if "apple" in thislist:

print("Yes, 'apple' is in the fruits list")

#-------------------------------------------------------

Python - Change List Items

Change Item Value

To change the value of a specific item, refer to the index number:

Change the second item:

thislist = ["apple", "banana", "cherry"]

thislist[1] = "blackcurrant"

print(thislist)

output:

['apple', 'blackcurrant', 'cherry']

#----------------------------------------

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of

index numbers

where you want to insert the new values:

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]

thislist[1:3] = ["blackcurrant", "watermelon"]

print(thislist)

output:

['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']

#---------------------------------------------------------

If you insert more items than you replace, the new items will be inserted where you specified,

and the remaining items will move accordingly:

Change the second value by replacing it with two new values:

thislist = ["apple", "banana", "cherry"]

thislist[1:2] = ["blackcurrant", "watermelon"]

print(thislist)

output:

['apple', 'blackcurrant', 'watermelon', 'cherry']

#-----------------------------------------------------

Note: The length of the list will change when the number of

items inserted does not match the number

of items replaced.

If you insert less items than you replace, the new items will

be inserted where you specified, and the remaining items will

move accordingly:

Change the second and third value by replacing it with one value:

thislist = ["apple", "banana", "cherry"]

thislist[1:3] = ["watermelon"]

print(thislist)

output:

['apple', 'watermelon']

#------------------------------------------------------------

Insert Items

To insert a new list item, without replacing any of the existing

values, we can use the insert() method.

The insert() method inserts an item at the specified index:

Insert "watermelon" as the third item:

thislist = ["apple", "banana", "cherry"]

thislist.insert(2, "watermelon")

print(thislist)

output:

['apple', 'banana', 'watermelon', 'cherry']

#------------------------------------------------------------------------

Python - Add List Items

Append Items

To add an item to the end of the list, use the append() method:

Using the append() method to append an item:

thislist = ["apple", "banana", "cherry"]

thislist.append("orange")

print(thislist)

output:

['apple', 'banana', 'cherry', 'orange']

#--------------------------------------------------

Insert Items

To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

Insert an item as the second position:

thislist = ["apple", "banana", "cherry"]

thislist.insert(1, "orange")

print(thislist)

output:

['apple', 'orange', 'banana', 'cherry']

-----------------------------

thislist = ["apple", "banana", "cherry"]

pk=["jackfruit",'Mango']

thislist.insert(1, pk)

print(thislist)

output:

['apple', ['jackfruit', 'Mango'], 'banana', 'cherry']

#------------------------------------------------------------

Extend List

To append elements from another list to the current list, use

the extend() method.

Add the elements of tropical to thislist:

thislist = ["apple", "banana", "cherry"]

tropical = ["mango", "pineapple", "papaya"]

thislist.extend(tropical)

print(thislist)

output:

['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

The elements will be added to the end of the list.

#---------------------------------------------------

Add Any Iterable

The extend() method does not have to append lists, you can add any iterable object

(tuples, sets, dictionaries etc.).

Add elements of a tuple to a list:

thislist = ["apple", "banana", "cherry"]

thistuple = ("kiwi", "orange")

thislist.extend(thistuple)

print(thislist)

output:

['apple', 'banana', 'cherry', 'kiwi', 'orange']

#-----------------------------------------------

Python - Remove List Items

Remove Specified Item

The remove() method removes the specified item.

Remove "banana":

thislist = ["apple", "banana", "cherry"]

thislist.remove("banana")

print(thislist)

output:

['apple', 'cherry']

#---------------------------------------------

Remove Specified Index

The pop() method removes the specified index.

Remove the second item:

thislist = ["apple", "banana", "cherry"]

#thislist.pop(1)

print(thislist)

#thislist.pop(-1)# removes last element

output:

['apple', 'cherry']

If you do not specify the index, the pop() method removes

the last item.

#-------------------------------------------------

Remove the last item:

thislist = ["apple", "banana", "cherry"]

thislist.pop()

print(thislist)

output:

['apple', 'banana']

#---------------------------------------------------------

The del keyword also removes the specified index:

Remove the first item:

thislist = ["apple", "banana", "cherry"]

del thislist[0]

print(thislist)

output:

['banana', 'cherry']

#-------------------------------------------------

The del keyword can also delete the list completely.

Delete the entire list:

thislist = ["apple", "banana", "cherry"]

print(thislist)

del thislist

print(thislist) # error

#--------------------------------------------------

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

#----------------------------------------------------------

Clear the list content:

thislist = ["apple", "banana", "cherry"]

print(thislist)

thislist.clear()

print(thislist)

output:

['apple', 'banana', 'cherry']

[]

#----------------------------------------

Python - Loop Lists

Loop Through a List

You can loop through the list items by using a for loop:

Print all items in the list, one by one:

thislist = ["apple", "banana", "cherry"]

for x in thislist:

print(x)

output:

apple

banana

cherry

#-----------------------------------------------------

Loop Through the Index Numbers

You can also loop through the list items by referring to their

index number.

Use the range() and len() functions to create a suitable iterable.

Print all items by referring to their index number:

thislist = ["apple", "banana", "cherry"]

for i in range(len(thislist)):

print(thislist[i])

output:

apple

banana

cherry

#------------------------------------------------------------

Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

Print all items, using a while loop to go through all the index numbers

thislist = ["apple", "banana", "cherry"]

i = 0

while i < len(thislist):

print(thislist[i])

i = i + 1

output:

apple

banana

cherry

#----------------------------------------------------------------------

Python - Sort Lists

Sort List Alphanumerically

List objects have a sort() method that will sort the list

alphanumerically, ascending, by default:

Sort the list alphabetically:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

thislist.sort()

print(thislist)

output:

['banana', 'kiwi', 'mango', 'orange', 'pineapple']

#--------------------------------------------------------

Sort the list numerically:

thislist = [100, 50, 65, 82, 23]

thislist.sort()

print(thislist)

output:

[23, 50, 65, 82, 100]

#-----------------------------------

Sort Descending

To sort descending, use the keyword argument reverse = True:

Sort the list descending:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

thislist.sort(reverse = True)

print(thislist)

output:

['pineapple', 'orange', 'mango', 'kiwi', 'banana']

#--------------------------------------------

Sort the list descending:

thislist = [100, 50, 65, 82, 23]

thislist.sort(reverse = True)

print(thislist)

output:

[100, 82, 65, 50, 23]

#----------------------------------------

Reverse Order

What if you want to reverse the order of a list, regardless of

the alphabet?

The reverse() method reverses the current sorting order of the elements.

Reverse the order of the list items:

thislist = ["banana", "Orange", "Kiwi", "cherry"]

thislist.reverse()

print(thislist)

output:

['cherry', 'Kiwi', 'Orange', 'banana']

#---------------------------------------------------------------------

Python - Copy Lists

Copy a List

You cannot copy a list simply by typing list2 = list1, because:

list2 will only be a reference to list1,

and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

Make a copy of a list with the copy() method:

thislist = ["apple", "banana", "cherry"]

mylist = thislist.copy()

print(mylist)

output:

['apple', 'banana', 'cherry']

#-------------------------------------------------

Another way to make a copy is to use the built-in method list().

Make a copy of a list with the list() method:

thislist = ["apple", "banana", "cherry"]

mylist = list(thislist)

print(mylist)

output:

['apple', 'banana', 'cherry']

#-------------------------------------------------

Python - Join Lists

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Join two list:

list1 = ["a", "b", "c"]

list2 = [1, 2, 3]

list3 = list1 + list2

print(list3)

output:

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

#---------------------------------------------------

Another way to join two lists is by appending all the items from list2 into list1, one by one:

Append list2 into list1:

list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

for x in list2:

list1.append(x)

print(list1)

output:

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

#-------------------------------------------------------

Or you can use the extend() method, which purpose is to add

elements from one list to another list:

Use the extend() method to add list2 at the end of list1:

list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

list1.extend(list2)

print(list1)

output:

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

#-----------------------------------------------

Python - List Comprehension

List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the

values of an existing list.

Without list comprehension you will have to write a for statement with a conditional test inside:

----------------------------------

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = []

for x in fruits:

if "a" in x:

newlist.append(x)

print(newlist)

output:

['apple', 'banana', 'mango']

#------------------------------------------------------------

With list comprehension you can do all that with only one line of code:

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]

print(newlist)

output:

['apple', 'banana', 'mango']

#----------------------------------------------------------

The Syntax

newlist = [expression for item in iterable if condition == True]

The return value is a new list, leaving the old list unchanged.

Condition

The condition is like a filter that only accepts the items that valuate to True.

Only accept items that are not "apple":

-------------------------

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if x != "apple"]

print(newlist)

output:

['banana', 'cherry', 'kiwi', 'mango']

------------------------------

The condition if x != "apple" will return True for all elements other than "apple", making the new list contain

all fruits except "apple".

The condition is optional and can be omitted:

#--------------------------------------------------------

With no if statement:

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits]

print(newlist)

output:

['apple', 'banana', 'cherry', 'kiwi', 'mango']

#--------------------------------------------------------

Iterable

The iterable can be any iterable object, like a list, tuple, set etc.

You can use the range() function to create an iterable:

newlist = [x for x in range(10)]

print(newlist)

output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

#--------------------------------------------------------------

Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]

print(newlist)

output:

[0, 1, 2, 3, 4]

#-------------------------------------------

**Expression**

The expression is the current item in the iteration, but it is also the outcome, which you can manipulate before

it ends up like a list item in the new list:

Set the values in the new list to upper case:

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x.upper() for x in fruits]

print(newlist)

output:

['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']

#---------------------------------------------------------------

Set all values in the new list to 'hello':

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = ['hello' for x in fruits]

print(newlist)

output:

['hello', 'hello', 'hello', 'hello', 'hello']

#------------------------------------------------------------

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#---------------------------------------------------------------------------

**Python Tuples**

Tuple

Tuples are used to store multiple items in a single

variable.

Tuple is one of 4 built-in data types in Python used to

store collections of data, the other

3 are List, Set, and Dictionary, all with different

qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

#----------------------------------------------------

#Create a Tuple:

thistuple = ("apple", "banana", "cherry")

print(thistuple)

output:

('apple', 'banana', 'cherry')

#---------------------------------------------------

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate

values.

Tuple items are indexed, the first item has index [0],

the second item has index [1] etc.

Ordered:

When we say that tuples are ordered, it means that the

items have a defined order, and that order

will not change.

Unchangeable:

Tuples are unchangeable, meaning that we cannot change,

add or remove items after the tuple has been created.

Allow Duplicates:

Since tuples are indexed, they can have items with the

same value:

#-------------------------------------------------------------

#Tuples allow duplicate values:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")

print(thistuple)

output:

('apple', 'banana', 'cherry', 'apple', 'cherry')

#-------------------------------------------------------------

Tuple Length

To determine how many items a tuple has, use the len()

function:

#Print the number of items in the tuple:

thistuple = ("apple", "banana", "cherry")

print(len(thistuple))

output:

3

#----------------------------------------------------------

Create Tuple With One Item

To create a tuple with only one item, you have to add a

comma after the item,

otherwise Python will not recognize it as a tuple.

One item tuple, remember the comma:

thistuple = ("apple",)

print(type(thistuple))

#NOT a tuple

thistuple = ("apple")

print(type(thistuple))

output:

<class 'tuple'>

<class 'str'>

#-------------------------------

Tuple Items - Data Types

Tuple items can be of any data type:

String, int and boolean data types:

tuple1 = ("apple", "banana", "cherry")

tuple2 = (1, 5, 7, 9, 3)

tuple3 = (True, False, False)

print(tuple1)

print(tuple2)

print(tuple3)

#--------------------------------------------------------

type()

From Python's perspective, tuples are defined as objects

with the data type 'tuple':

<class 'tuple'>

What is the data type of a tuple?

-------------------------------------

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

print(type(mytuple))

output:

<class 'tuple'>

#---------------------------------------------------------------------------

The tuple() Constructor

It is also possible to use the tuple() constructor to

make a tuple.

Using the tuple() method to make a tuple:

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets

print(thistuple)

thistuple2 = tuple(["apple", "banana", "cherry"]) # note the square brackets(list)

print(thistuple2)

output:

('apple', 'banana', 'cherry')

('apple', 'banana', 'cherry')

#--------------------------------------------------------------------------

Access Tuple Items

You can access tuple items by referring to the index

number, inside square brackets:

Print the second item in the tuple:

thistuple = ("apple", "banana", "cherry")

print(thistuple[1])

output:

banana

Note: The first item has index 0.

#----------------------------------------------------------------------

Print the last item of the tuple:

thistuple = ("apple", "banana", "cherry")

print(thistuple[-1])

output:

cherry

#---------------------------------------------------------------------

Range of Indexes

You can specify a range of indexes by specifying where

to start and where to end the range.

When specifying a range, the return value will be a

new tuple with the specified items.

Return the third, fourth, and fifth item:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print(thistuple[2:5])

output:

('cherry', 'orange', 'kiwi')

Note: The search will start at index 2 (included) and

end at index 5 (not included).

#-----------------------------------------------------------------------

By leaving out the start value, the range will start at

the first item:

returns the items from the beginning to, but NOT included, "kiwi":

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print(thistuple[:4])

output:

('apple', 'banana', 'cherry', 'orange')

#---------------------------------------------------------------------

By leaving out the end value, the range will go on to

the end of the list:

returns the items from "cherry" and to the end:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print(thistuple[2:])

output:

('cherry', 'orange', 'kiwi', 'melon', 'mango')

#-----------------------------------------------------------------------

Range of Negative Indexes

Specify negative indexes if you want to start the search

from the end of the tuple:

returns the items from index -4 (included) to index

-1 (excluded)

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")

print(thistuple[-4:-1])

output:

('orange', 'kiwi', 'melon')

#------------------------------------------------------------------------

Check if Item Exists

To determine if a specified item is present in a tuple

use the in keyword:

Check if "apple" is present in the tuple:

thistuple = ("apple", "banana", "cherry")

if "apple" in thistuple:

print("Yes, 'apple' is in the fruits tuple")

output:

Yes, 'apple' is in the fruits tuple

#---------------------------------------------------------------------

Update Tuples

Tuples are unchangeable, meaning that you cannot change,

add, or remove items once the tuple

is created.

But there are some workarounds.

Change Tuple Values

Once a tuple is created, you cannot change its values.

Tuples are unchangeable, or immutable as

it also is called.

But there is a workaround. You can convert the tuple

into a list, change the list, and convert

the list back into a tuple.

Convert the tuple into a list to be able to change it:

x = ("apple", "banana", "cherry")

y = list(x)

y[1] = "kiwi"

x = tuple(y)

print(x)

output:

('apple', 'kiwi', 'cherry')

#-------------------------------------------------------------------------------

Add Items

Since tuples are immutable, they do not have a build-in

append() method, but there are

other ways to add items to a tuple.

1. Convert into a list: Just like the workaround for

changing a tuple, you can convert it

into a list, add your item(s), and convert it back into

a tuple.

Convert the tuple into a list, add "orange", and convert it back into a tuple:

thistuple = ("apple", "banana", "cherry")

y = list(thistuple)

y.append("orange")

thistuple = tuple(y)

print(thistuple)

output:

('apple', 'banana', 'cherry', 'orange')

#---------------------------------------------------------------------------------------------------

2. Add tuple to a tuple. You are allowed to add tuples

to tuples, so if you want to add one item, (or many),

create a new tuple with the item(s), and add it to the

existing tuple:

Create a new tuple with the value "orange", and add

that tuple:

thistuple = ("apple", "banana", "cherry")

y = ("orange",)

thistuple += y

print(thistuple)

output:

('apple', 'banana', 'cherry', 'orange')

#---------------------------------------------------------------------------------------

Remove Items

Note: You cannot remove items in a tuple.

Tuples are unchangeable, so you cannot remove items

from it, but you can use the same workaround as

we used for changing and adding tuple items:

Convert the tuple into a list, remove "apple", and

convert it back into a tuple:

thistuple = ("apple", "banana", "cherry")

y = list(thistuple)

y.remove("apple")

thistuple = tuple(y)

print(thistuple)

output:

('banana', 'cherry')

#-----------------------------------------------------------------------------

can delete the tuple completely:

The del keyword can delete the tuple completely:

thistuple = ("apple", "banana", "cherry")

del thistuple

print(thistuple) #this will raise an error because the

tuple no longer exists

#-----------------------------------------------------------------------------

Python - Unpack Tuples

Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

Packing a tuple:

fruits = ("apple", "banana", "cherry")

print(fruits)

output:

('apple', 'banana', 'cherry')

#--------------------------------------------------------------------------------------

But, in Python, we are also allowed to extract the values

back into variables. This is called

"unpacking":

Unpacking a tuple:

----------------------------

fruits = ("apple", "banana", "cherry")

(green, yellow, red) = fruits

print(green)

print(yellow)

print(red)

output:

apple

banana

cherry

-----------------------------------

Note: The number of variables must match the number

of values in the tuple, if not,

you must use an asterisk to collect the remaining

values as a list.

#------------------------------------------------------------------------------------

Using Asterisk\*

If the number of variables is less than the number of

values, you can add an \* to the

variable name and the values will be assigned to the

variable as a list:

Assign the rest of the values as a list called "red":

-----------------------

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, \*red) = fruits

print(green)

print(yellow)

print(red)

output:

apple

banana

['cherry', 'strawberry', 'raspberry']

#------------------------------------------------------------------------------

If the asterisk is added to another variable name than

the last, Python will assign values to

the variable until the number of values left matches

the number of variables left.

Add a list of values the "tropic" variable:

fruits = ("apple", "mango", "papaya", "pineapple", "cherry")

(green, \*tropic, red) = fruits

print(green)

print(tropic)

print(red)

output:

apple

['mango', 'papaya', 'pineapple']

cherry

#-----------------------------------------------------------------------------

Loop Through a Tuple

You can loop through the tuple items by using a for loop.

Iterate through the items and print the values:

thistuple = ("apple", "banana", "cherry")

for x in thistuple:

print(x)

output:

apple

banana

cherry

#-------------------------------------------------------------------------------

Loop Through the Index Numbers

You can also loop through the tuple items by referring

to their index number.

Use the range() and len() functions to create a suitable

iterable.

Print all items by referring to their index number:

thistuple = ("apple", "banana", "cherry")

for i in range(len(thistuple)):

print(thistuple[i])

output:

apple

banana

cherry

#-----------------------------------------------------------------------------

Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the

tuple, then start at 0 and loop

your way through the tuple items by refering to their

indexes.

Remember to increase the index by 1 after each iteration.

Print all items, using a while loop to go through all

the index numbers:

thistuple = ("apple", "banana", "cherry")

i = 0

while i < len(thistuple):

print(thistuple[i])

i = i + 1

output:

apple

banana

cherry

#--------------------------------------------------------------------------------

Python - Join Tuples

Join Two Tuples

To join two or more tuples you can use the + operator:

Join two tuples:

tuple1 = ("a", "b" , "c")

tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2

print(tuple3)

output:

('a', 'b', 'c', 1, 2, 3)

#--------------------------------------------------------------------------

Multiply Tuples

If you want to multiply the content of a tuple a given

number of times, you can use

the \* operator:

Multiply the fruits tuple by 2:

fruits = ("apple", "banana", "cherry")

mytuple = fruits \* 2

print(mytuple)

output:

('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')

#-------------------------------------------------------------------------------------------

Python - Tuple Methods

Python Tuple count() Method

Return the number of times the value 5 appears in

the tuple:

thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)

output:

2

#-------------------------------------------------------------------------

Python Tuple index() Method

Search for the first occurrence of the value 8, and return its position:

thistuple = (10, 30, 70, 80, 70, 50, 40, 60, 80, 50)

x = thistuple.index(80)

print(x)

output:

3

#------------------------------------------------------------------------

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#---------------------------------------------------------

**Python Sets**

Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of

data, the other 3 are List,

Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable\*, and unindexed.

\* Note: Set items are unchangeable, but you can remove items and add new items

Sets are written with curly brackets.

#--------------------------------------------------------------------------------

Create a Set:

thisset = {"apple", "banana", "cherry"}

print(thisset)

output:

{'apple', 'banana', 'cherry'}

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

#--------------------------------------------------------------------------

Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

Duplicates Not Allowed

Sets cannot have two items with the same value.

#----------------------------------------------------------

Duplicate values will be ignored:

thisset = {"apple", "banana", "cherry

output:", "apple"}

print(thisset)

{'apple', 'cherry', 'banana'}

#-----------------------------------------------------------

Get the Length of a Set

To determine how many items a set has, use the len() function.

Get the number of items in a set:

thisset = {"apple", "banana", "cherry"}

print(len(thisset))

output:

3

#---------------------------------------------------------------------

Set Items - Data Types

Set items can be of any data type:

String, int and boolean data types:

---------------------

set1 = {"apple", "banana", "cherry"}

set2 = {1, 5, 7, 9, 3}

set3 = {True, False, False}

print(set1)

print(set2)

print(set3)

output:

{'banana', 'apple', 'cherry'}

{1, 3, 5, 7, 9}

{False, True}

#----------------------------------------------------------

A set can contain different data types:

A set with strings, integers and boolean values:

set1 = {"abc", 34, True, 40, "male"}

print(set1)

output:

{True, 34, 40, 'abc', 'male'}

#---------------------------------------------------------------

type()

From Python's perspective, sets are defined as objects with the data type 'set':

<class 'set'>

What is the data type of a set?

--------------------

myset = {"apple", "banana", "cherry"}

print(type(myset))

output:

<class 'set'>

#------------------------------------------------------------------

The set() Constructor

It is also possible to use the set() constructor to make a set.

Using the set() constructor to make a set:

---------------------

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets

print(thisset)

output:

{'apple', 'banana', 'cherry'}

#-----------------------------------------------------------------------------

Python - Access Set Items

Access Items

You cannot access items in a set by referring to an

index or a key.

But you can loop through the set items using a for

loop, or ask if a specified value is present in a set,

by using the in keyword.

Loop through the set, and print the values:

--------------------------

thisset = {"apple", "banana", "cherry"}

for x in thisset:

print(x)

output:

banana

apple

cherry

#----------------------------------------------------------------------------

Check if "banana" is present in the set:

-------------------

thisset = {"apple", "banana", "cherry"}

print("banana" in thisset)

output:

True

------------------------

Change Items

Once a set is created, you cannot change its items,

but you can add new items.

#---------------------------------------------------------------------------------

Python - Add Set Items

Add Items

Once a set is created, you cannot change its items,

but you can add new items.

To add one item to a set use the add() method.

Add an item to a set, using the add() method:

-----------------------------

thisset = {"apple", "banana", "cherry"}

thisset.add("orange")

print(thisset)

output:

{'orange', 'cherry', 'banana', 'apple'}

#-------------------------------------------------------------------------

Add Sets

To add items from another set into the current set,

use the update() method.

Add elements from tropical into thisset:

-------------------------

thisset = {"apple", "banana", "cherry"}

tropical = {"pineapple", "mango", "papaya"}

thisset.update(tropical)

print(thisset)

output:

{'pineapple', 'cherry', 'banana', 'mango', 'papaya', 'apple'}

#-------------------------------------------------------------------

Add Any Iterable

The object in the update() method does not have to be

a set, it can be any iterable object

(tuples, lists, dictionaries etc.).

Add elements of a list to at set:

-------------------------

thisset = {"apple", "banana", "cherry"}

mylist = ["kiwi", "orange"]

thisset.update(mylist)

print(thisset)

output:

{'apple', 'banana', 'cherry', 'kiwi', 'orange'}

#----------------------------------------------------------------

Python - Remove Set Items

Remove Item

To remove an item in a set, use the remove(), or the

discard() method.

Remove "banana" by using the remove() method:

-------------------

thisset = {"apple", "banana", "cherry"}

thisset.remove("banana")

print(thisset)

output:

{'cherry', 'apple'}

----------------------

Note: If the item to remove does not exist, remove()

will raise an error.

#------------------------------------------------------------------

Remove "banana" by using the discard() method:

------------------

thisset = {"apple", "banana", "cherry"}

thisset.discard("banana")

print(thisset)

output:

{'apple', 'cherry'}

-----------------

Note: If the item to remove does not exist, discard()

will NOT raise an error.

#-----------------------------------------------------------------

You can also use the pop() method to remove an item,

but this method will remove the last item.

Remember that sets are unordered, so you will not

know what item that gets removed.

The return value of the pop() method is the removed item

-----------------------

thisset = {"apple", "banana", "cherry"}

x = thisset.pop()

print(x)

print(thisset)

output:

cherry

{'banana', 'apple'}

---------------

Note: Sets are unordered, so when using the pop() method, you do not know which item that gets removed.

#----------------------------------------------------------------------------------

The clear() method empties the set:

thisset = {"apple", "banana", "cherry"}

thisset.clear()

print(thisset)

output:

set()

#--------------------------------------------------------

The del keyword will delete the set completely:

thisset = {"apple", "banana", "cherry"}

del thisset

print(thisset)

#-----------------------------------------------------------

Python - Loop Sets

Loop Items

You can loop through the set items by using a for loop:

Loop through the set, and print the values:

----------------------

thisset = {"apple", "banana", "cherry"}

for x in thisset:

print(x)

#--------------------------------------------------------------

Python Set copy() Method

Copy the fruits set:

-------------------

fruits = {"apple", "banana", "cherry"}

x = fruits.copy()

print(x)

#-----------------------------------------------------------------

Join Two Sets

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets

The union() method returns a new set with all items from both sets:

#------------------------------------------------------------------

The update() method inserts the items in set2 into set1:

set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}

set1.update(set2)

print(set1)

output:

{'c', 1, 2, 3, 'a', 'b'}

#---------------------------------------------------------

Python Set Operations

Set can be performed mathematical operation such as union, intersection, difference, and symmetric difference.

Python provides the facility to carry out these operations with operators or methods.

Union of two Sets

The union of two sets is calculated by using the pipe (|) operator. The union of the two sets contains all the

items that are present in both the sets.

to calculate the union of two sets.

1: using union | operator

Days1 = {"Monday","Tuesday","Wednesday","Thursday", "Sunday"}

Days2 = {"Friday","Saturday","Sunday"}

print(Days1|Days2) #printing the union of the sets

output:

{'Wednesday', 'Tuesday', 'Thursday', 'Saturday', 'Friday', 'Monday', 'Sunday'}

-------------------

Python also provides the union() method which can also be used to calculate the union of two sets.

2: using union() method

Days1 = {"Monday","Tuesday","Wednesday","Thursday"}

Days2 = {"Friday","Saturday","Sunday"}

print(Days1.union(Days2)) #printing the union of the sets

output:

{'Tuesday', 'Wednesday', 'Sunday', 'Friday', 'Saturday', 'Monday', 'Thursday'}

#---------------------------------------------------------------------

Intersection of two sets

The intersection of two sets can be performed by the and & operator or the intersection() function.

The intersection of the two sets is given as the set of the elements that common in both sets.

1: Using & operator

Days1 = {"Monday","Tuesday", "Wednesday", "Thursday"}

Days2 = {"Monday","Tuesday","Sunday", "Friday"}

print(Days1&Days2) #prints the intersection of the two sets

output:

{'Tuesday', 'Monday'}

---------------------

2: Using intersection() method

set1 = {"Devansh","John", "David", "Martin"}

set2 = {"Steve", "Milan", "David", "Martin"}

print(set1.intersection(set2)) #prints the intersection of the two sets

output:

{'Martin', 'David'}

-----------------------

3:Using intersection() method

set1 = {1,2,3,4,5,6,7}

set2 = {1,2,20,32,5,9}

set3 = set1.intersection(set2)

print(set3)

output:

{'Martin', 'David'}

#--------------------------------------------------------------

Difference between the two sets

The difference of two sets can be calculated by using the subtraction (-) operator or intersection() method.

Suppose there are two sets A and B, and the difference is A-B that denotes the resulting set will be

obtained that element of A, which is not present in the set B.

1 : Using subtraction ( - ) operator

-----------------------------

Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}

Days2 = {"Monday", "Tuesday", "Sunday"}

print(Days1-Days2) #{"Wednesday", "Thursday" will be printed}

output:

{'Wednesday', 'Thursday'}

---------------

2 : Using difference() method

----------------------------

Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}

Days2 = {"Monday", "Tuesday", "Sunday"}

print(Days1.difference(Days2)) # prints the difference of the two sets Days1 and Days2

output:

{'Wednesday', 'Thursday'}

#-----------------------------------------------------------------------------------------

Symmetric Difference of two sets

The symmetric difference of two sets is calculated by ^ operator or symmetric\_difference() method.

Symmetric difference of sets, it removes that element which is present in both sets.

1: Using ^ operator

a = {1,2,3,4,5,6}

b = {1,2,9,8,10}

c = a^b

print(c)

output:

{3, 4, 5, 6, 8, 9, 10}

------------------------

2: Using symmetric\_difference() method

a = {1,2,3,4,5,6}

b = {1,2,9,8,10}

c = a.symmetric\_difference(b)

print(c)

output:

{3, 4, 5, 6, 8, 9, 10}

#------------------------------------------------------------------------------

Set comparisons

Python allows us to use the comparison operators i.e.,

<, >, <=, >= , == with the sets by using which we can

check whether a set is a subset, superset, or equivalent

to other set. The boolean true or false

is returned depending upon the items present inside

the sets.

------------------

Days1 = {"Monday", "Tuesday", "Wednesday", "Thursday"}

Days2 = {"Monday", "Tuesday"}

Days3 = {"Monday", "Tuesday", "Friday"}

#Days1 is the superset of Days2 hence it will print true.

print (Days1>Days2)

#prints false since Days1 is not the subset of Days2

print (Days1<Days2)

#prints false since Days2 and Days3 are not equivalent

print (Days2 == Days3)

output:

True

False

False

#----------------------------------------------------------------------

Python Set issubset() Method

Return True if all items in set x are present in set y:

x = {"a", "b", "c"}

y = {"f", "e", "d", "c", "b", "a"}

z = x.issubset(y)

print(z)

output:

True

#---------------------------------------------------------------------

Python Set issuperset() Method

Return True if all items set y are present in set x:

x = {"f", "e", "d", "c", "b", "a"}

y = {"a", "b", "c"}

z = x.issuperset(y)

print(z)

output:

True

#-------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--------------------------------------------

**Python Dictionaries**

**Dictionary**

Dictionaries are used to store data values in key:value pairs.

A dictionary is a collection which is ordered\*, changeable and do not allow

duplicates.

Dictionaries are written with curly brackets, and have keys and values:

#---------------------------------------------------------------

Create and print a dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

print(thisdict)

output:

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

#-----------------------------------------------

Dictionary Items

Dictionary items are ordered, changeable, and does not allow duplicates.

Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

#--------------------------------------------------------------------------------------------

Print the "brand" value of the dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

print(thisdict["brand"])

output:

Ford

#-----------------------------------------------------------------

Ordered or Unordered?

When we say that dictionaries are ordered, it means that the items have a defined order,

and that order will not change.

Unordered means that the items does not have a defined order, you cannot refer to an

item by using an index.

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the

dictionary has been created.

Duplicates Not Allowed

Dictionaries cannot have two items with the same key:

Duplicate values will overwrite existing values:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964,

"year": 2020

}

print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

#---------------------------------------------------------------------------------

Dictionary Length

To determine how many items a dictionary has, use the len() function:

Print the number of items in the dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964,

"year": 2020

}

print(len(thisdict))

output:

3

#-----------------------------------------------

Dictionary Items - Data Types

The values in dictionary items can be of any data type:

String, int, boolean, and list data types:

thisdict = {

"brand": "Ford",

"electric": False,

"year": 1964,

"colors": ["red", "white", "blue"]

}

print(thisdict)

output:

{'brand': 'Ford', 'electric': False, 'year': 1964, 'colors': ['red', 'white', 'blue']}

#-------------------------------------------------------------------------------

type()

From Python's perspective, dictionaries are defined as objects with the data type 'dict':

<class 'dict'>

Print the data type of a dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

print(type(thisdict))

output:

<class 'dict'>

#------------------------------------------------------------------

Python - Access Dictionary Items

Accessing Items

Get the value of the "model" key:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict["model"]

print(x)

output:

Mustang

#--------------------------------------------------------------

There is also a method called get() that will give you the same result:

Get the value of the "model" key:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict.get("model")

print(x)

output:

Mustang

#---------------------------------------------------------------

Get Keys

The keys() method will return a list of all the keys in the

dictionary.

Get a list of the keys:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict.keys()

print(x)

output:

dict\_keys(['brand', 'model', 'year'])

#------------------------------------------------------------

Add a new item to the original dictionary, and see that

the keys list gets updated as well:

car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change

output:

dict\_keys(['brand', 'model', 'year'])

dict\_keys(['brand', 'model', 'year', 'color'])

#-------------------------------------------------------------------

Get Values

The values() method will return a list of all the values

in the dictionary.

Get a list of the values:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict.values()

print(x)

output:

dict\_values(['Ford', 'Mustang', 1964])

#--------------------------------------------------------------

Make a change in the original dictionary, and see that

the values list gets updated as well:

car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = car.values()

print(x) #before the change

car["year"] = 2020

print(x) #after the change

output:

dict\_values(['Ford', 'Mustang', 1964])

dict\_values(['Ford', 'Mustang', 2020])

#------------------------------------------------------------------------

Add a new item to the original dictionary, and see that

the values list gets updated as well:

car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = car.values()

print(x) #before the change

car["color"] = "red"

print(x) #after the change

output:

dict\_values(['Ford', 'Mustang', 1964])

dict\_values(['Ford', 'Mustang', 1964, 'red'])

#----------------------------------------------------------------------

Get Items

The items() method will return each item in a dictionary,

as tuples in a list

Get a list of the key:value pairs

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = thisdict.items()

print(x)

output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

#---------------------------------------------------------------------------

Make a change in the original dictionary, and see that the

items list gets updated as well:

car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = car.items()

print(x) #before the change

car["year"] = 2020

print(x) #after the change

output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2020)])

#---------------------------------------------------

Add a new item to the original dictionary, and see that

the items list gets updated as well:

car = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

x = car.items()

print(x) #before the change

car["color"] = "red"

print(x) #after the change

output:

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964), ('color', 'red')])

#------------------------------------------------------------------------------

Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword:

Check if "model" is present in the dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

if "model" in thisdict:

print("Yes, 'model' is one of the keys in the thisdict dictionary")

#-------------------------------------------------------------------------------

Change Values

You can change the value of a specific item by referring to its key name:

Change the "year" to 2018:

------------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict["year"] = 2018

print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 2018}

#----------------------------------------------------------------------------

Update Dictionary

The update() method will update the dictionary with the

items from the given argument.

The argument must be a dictionary, or an iterable object

with key:value pairs.

Update the "year" of the car by using the update() method:

-------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict.update({"year": 2020})

print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

-------------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict.update({"year": 2020})

print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 2020}

#------------------------------------------------------------------

Python - Add Dictionary Items

Adding Items

Adding an item to the dictionary is done by using a new

index key and assigning a value to it:

------------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict["color"] = "red"

print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

#-------------------------------------------------------------------

Update Dictionary

The update() method will update the dictionary with the items from a given argument. If the item does not exist,

the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

Add a color item to the dictionary by using the update() method:

---------------------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict.update({"color": "red"})

print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

#-------------------------------------------------------------------------------------

Python - Remove Dictionary Items

Removing Items

There are several methods to remove items from a dictionary:

The pop() method removes the item with the specified key name:

----------------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict.pop("model")

print(thisdict)

output:

{'brand': 'Ford', 'year': 1964}

#-----------------------------------------------------------------------------

The popitem() method removes the last inserted item

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict.popitem()

print(thisdict)

output:

{'brand': 'Ford', 'model': 'Mustang'}

#-----------------------------------------------------------

The del keyword removes the item with the specified key name:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

del thisdict["model"]

print(thisdict)

output:

{'brand': 'Ford', 'year': 1964}

#-------------------------------------------------------------------------

The del keyword can also delete the dictionary completely:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

del thisdict

print(thisdict) #this will cause an error because

"thisdict" no longer exists.

#------------------------------------------------------------------------------------

The clear() method empties the dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

thisdict.clear()

print(thisdict)

output:

{}

#--------------------------------------------------

Python - Loop Dictionaries

Loop Through a Dictionary

You can loop through a dictionary by using a for loop.

When looping through a dictionary, the return value are

the keys of the dictionary,

but there are methods

to return the values as well.

Print all key names in the dictionary, one by one:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

for x in thisdict:

print(x)

output:

brand

model

year

#-----------------------------------------------------------------

Print all values in the dictionary, one by one:

----------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

for x in thisdict:

print(thisdict[x])

output:

Ford

Mustang

1964

-----------------------

You can also use the values() method to return values of .

a dictionary:

------------------

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

for x in thisdict.values():

print(x)

output:

Ford

Mustang

1964

#----------------------------------------------------------------

You can use the keys() method to return the keys of a

dictionary:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

for x in thisdict.keys():

print(x)

#--------------------------------------------------------------------

Loop through both keys and values, by using the items() method:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

for x, y in thisdict.items():

print(x, y)

output:

brand Ford

model Mustang

year 1964

#-------------------------------------------------------------------

Python - Copy Dictionaries

Copy a Dictionary

You cannot copy a dictionary simply by typing dict2 = dict1,

because: dict2 will only be a reference to dict1, and

changes made in dict1 will automatically also be made

in dict2.

There are ways to make a copy, one way is to use the

built-in Dictionary method copy().

Make a copy of a dictionary with the copy() method:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

mydict = thisdict.copy()

print(mydict)

output:

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

#-------------------------------------------------------------------------------

Make a copy of a dictionary with the dict() function:

thisdict = {

"brand": "Ford",

"model": "Mustang",

"year": 1964

}

mydict = dict(thisdict)

print(mydict)

output:

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

#--------------------------------------------------------------------------

Python - Nested Dictionaries

Nested Dictionaries

A dictionary can contain dictionaries, this is called

nested dictionaries.

Create a dictionary that contain three dictionaries:

--------------------

myfamily = {

"child1" : {

"name" : "Emil",

"year" : 2004

},

"child2" : {

"name" : "Tobias",

"year" : 2007

},

"child3" : {

"name" : "Linus",

"year" : 2011

}

}

print(myfamily)

output:

{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}

#-----------------------------------------------------------------------

Create three dictionaries, then create one dictionary that

will contain the

other three dictionaries:

child1 = {

"name" : "Emil",

"year" : 2004

}

child2 = {

"name" : "Tobias",

"year" : 2007

}

child3 = {

"name" : "Linus",

"year" : 2011

}

myfamily = {

"child1" : child1,

"child2" : child2,

"child3" : child3

}

print(myfamily)

output:

{'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}, 'child3': {'name': 'Linus', 'year': 2011}}

#--------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#----------------------------------------------------

**Python Function**

Functions are the most important aspect of an application.

A function can be defined

as the organized block of reusable

code, which can be called whenever required.

Python allows us to divide a large program into the basic

building blocks known as a function. The function contains

the set of programming statements enclosed by {}. A function can be called multiple times

to provide reusability and modularity to the Python program.

Functions are the most important aspect of an application.

A function can be defined as the organized block of

reusable code, which can be called whenever required.

The function contains the set of programming statements

enclosed by {}. A function can be called multiple times

to provide reusability and modularity to the Python

program. The Function helps to programmer to break the

program into the smaller part. It organizes

the code very effectively and avoids the repetition of

the code. As the program grows,

function makes the program more organized.

Python provide us various inbuilt functions like range()

or print(). Although, the user can create its functions,

which can be called user-defined functions.

There are mainly two types of functions.

User-define functions: - The user-defined functions are

those define by the user to perform the specific task.

Built-in functions: - The built-in functions are those

functions that are pre-defined in Python.

Advantage of Functions in Python

There are the following advantages of Python functions.

Using functions, we can avoid rewriting the same logic/code

again and again in a program. We can call Python functions

multiple times in a program and anywhere in a program.

We can track a large Python program easily when it is

divided into multiple functions. Reusability is the main

achievement of Python functions.

------------------

Creating a Function

Python provides the def keyword to define the function.

The syntax of the define function is given below.

Syntax:

def my\_function(parameters):

function\_block

[return expression]

Let's understand the syntax of functions definition.

The def keyword, along with the function name is used to

define the function. The identifier rule must follow the

function name. A function accepts the parameter (argument),

and they can be optional. The function block is started

with the colon (:), and block statements must be at the

same indentation.

The return statement is used to return the value. A

function can have only one return Function Calling

In Python, after the function is created, we can call it

from another function. A function must be defined before

the function call; otherwise, the Python interpreter

gives an error. To call the function,

use the function name followed

by the parentheses.

#----------------------------------------

#function definition

def hello():

print("hello world...bye")

# function calling

hello()

output:

hello world...bye

-----------------------------

The return statement:

The return statement is used at the end of the function

and returns the result of the function.

It terminates the function execution and transfers the

result where the function is called.

The return statement cannot be used outside of the function.

Syntax:

return [expression\_list]

It can contain the expression which gets evaluated and

value is returned to the caller function.

If the return statement has no expression or does not

exist itself in the function then it

returns the object.

----------------

# Defining function

def sum():

a = 10

b = 20

c = a+b

return c

# calling sum() function in print statement

print("The sum is:",sum())

output:

The sum is: 30

----------------------------------------------------

Creating function without return statement

----------

# Defining function

def sum():

a = 10

b = 20

c = a+b

# calling sum() function in print statement

print(sum())

output:

None

------------------------------

In the above code, we have defined the same function without the return statement as

we can see that

the sum() function returned the None object to the caller function.

Arguments in function

The arguments are types of information which can be passed into the function.

The arguments are

specified in the parentheses. We can pass any number of arguments, but they must

be separate them with a comma.

-------------------

# Python function to calculate the sum of two variables

# defining the function

def sum(a, b):

return a + b

# taking values from the user

a = int(input("Enter a: "))

b = int(input("Enter b: "))

# printing the sum of a and b

print("Sum = ", sum(a, b))

output:

Enter a: 20

Enter b: 30

Sum = 50

-----------------------------------------------------

Passing mutable Object (List)

# defining the function

def change\_list(list1):

list1.append(20)

list1.append(30)

print("list inside function = ", list1)

# defining the list

list1 = [10, 30, 40, 50]

# calling the function

change\_list(list1)

print("list outside function = ", list1)

output:

list inside function = [10, 30, 40, 50, 20, 30]

list outside function = [10, 30, 40, 50, 20, 30]

------------------------------------------------------

Passing Imutable Object (String)

------------

# defining the function

def change\_string(str):

str = str + " Hows you "

print("printing the string inside function :", str)

string1 = "Hi I am there"

# calling the function

change\_string(string1)

print("printing the string outside function :", string1)

output:

printing the string inside function : Hi I am there Hows you

printing the string outside function : Hi I am there

#----------------------------------------------------------

**Types of arguments**

There may be several types of arguments which can be passed

at the time of function call.

1. Required arguments

2. Keyword arguments

3. Default arguments

4. Variable-length arguments

We can provide the arguments at the time of the function

call. As far as the required arguments are concerned,

these are the arguments which are required to be passed

at the time of function calling with the exact match of

their positions in the function call and function

definition. If either of the arguments is not provided

in the function call, or the position of the arguments

is changed, the Python interpreter

will show the error.

#---------------------------------------------------------------------

1. Required arguments

def func(name):

message= "Hi "+name

return message

name = input("Enter name:")

print(func(name))

output:

Enter name:Amal

Hi Amal

#-----------------------------------------------------------

#the function simple\_interest accepts three arguments and

returns the simple interest accordingly

------------

def simple\_interest(p,t,r):

return (p\*t\*r)/100

p1 = float(input("Enter the principle amount:"))

r1 = float(input("Enter the rate of interest:"))

t1 = float(input("Enter the time in years:"))

print("Simple Interest: ",simple\_interest(p1,t1,r1))

output:

Enter the principle amount:10000

Enter the rate of interest:7

Enter the time in years:5

Simple Interest: 3500.0

#-----------------------------------------------------

#the function calculate returns the sum of two arguments a

and b

---------------------

def calculate(a,b):

return a+b

print(calculate(3,5))

print(calculate(10)) # error

# this causes an error as we are missing a required arguments b.

#-----------------------------------------------------------------

Default Arguments

Python allows us to initialize the arguments at the

function definition. If the value of any of the

arguments is not provided at the time of function call,

then that argument can be initialized with the value

given in the definition even if the argument is not

specified at the function call.

#--------------------------------------------------------

def printme(name,age=22):

print("My name is",name,"and age is",age)

printme(name = "john") #the variable age is not passed

output:

My name is john and age is 22

--------------------------

into the function however the default

value of age is considered in the function

printme(age = 10,name="David") #the value of age

is overwritten here, 10 will be printed as age

#-----------------------------------------------------------------------------

Variable-length Arguments (\*args)

In large projects, sometimes we may not know the number

of arguments to be passed in advance. In such cases,

Python provides us the flexibility to offer the

comma-separated values which are internally treated

as tuples at the function call.

By using the variable-length arguments, we can pass

any number of arguments.

However,at the function definition, we define the variable-length argument using

the \*args (star) as \*<variable - name >.

#----------------------------------------------------------------------

def printme(\*names):

for name in names:

print(name)

printme("john","David","smith","nick")

printme(10,20)

output:

john

David

smith

nick

10

20

#----------------------------------------------------------------------------------------------

In the above code, we passed \*names as variable-length

argument. We called the function and passed values

which are treated as tuple internally. The tuple is an

iterable sequence the same as the list. To print the

given values, we iterated \*arg names using for loop.

#-------------------------------------------------

Keyword arguments(\*\*kwargs)

Python allows us to call the function with the keyword

arguments. This kind of function call will enable us

to pass the arguments in the random order.

The name of the arguments is treated as the keywords

and matched in the function calling and definition.

If the same match is found, the values of the arguments

are copied in the function definition.

#-----------------------------------------------------------------------------------------------------

#function func is called with the name and message as the keyword arguments

def func(name,message):

print("printing the message with",name,"and ",message)

#name and message is copied with the values John and hello respectively

func(name = "John",message="hello")

output:

printing the message with John and hello

#---------------------------------------------------------------

Providing the values in different order at the calling

------------

#The function simple\_interest(p, t, r) is called with the keyword arguments the order

of arguments doesn't matter in this case

def simple\_interest(p,t,r):

return (p\*t\*r)/100

print("Simple Interest: ",simple\_interest(t=10,r=10,p=1900))

#----------------------------------------------------------------------------

If we provide the different name of arguments at the time of function call,

an error will be thrown.

Consider the following example.

--------------------

#The function simple\_interest(p, t, r) is called with the keyword arguments.

def simple\_interest(p,t,r):

return (p\*t\*r)/100

print("Amt:",simple\_interest(t=5,p=25000,r=7))

output:

Amt: 8750.0

-----------------------

# doesn't find the exact match of the name of the arguments (keywords)

print("Simple Interest: ",simple\_interest(time=10,rate=10,principle=1900))

#----------------------------------------------------------------------------

The Python allows us to provide the mix of the required

arguments and keyword arguments at the time of function

call. However, the required argument must not be given

after the keyword argument,

i.e., once the keyword argument is encountered in the

function call, the following arguments must also be

the keyword arguments.

-------------------

def func(name1,message,name2):

print("printing the message with",name1,",",message,",and",name2)

#the first argument is not the keyword argument

func("John",message="hello",name2="David")

output:

printing the message with John , hello ,and David

#------------------------------------------------------------------------

The following example will cause an error due to an

in-proper mix of keyword and required arguments being

passed in the function call.

------------------------

def func(name1,message,name2):

print("printing the message with",name1,",",message,",and",name2)

func("John",message="hello","David") #error

#------------------------------------------------------------------------

Python provides the facility to pass the multiple

keyword arguments which can be represented as \*\*kwargs.

It is similar as the \*args but it stores the argument

in the dictionary format.

This type of arguments is useful when we do not know

the number of arguments in advance.

Many arguments using Keyword argument

-------------------

def food(\*\*kwargs):

print(kwargs)

food(a="Apple")

food(fruits="Orange", Vagitables="Carrot")

output:

{'a': 'Apple'}

{'fruits': 'Orange', 'Vagitables': 'Carrot'}

#-----------------------------------------------------------------------------

Scope of variables

The scopes of the variables depend upon the location

where the variable is being declared.

The variable declared in one part of the program may

not be accessible to the other parts.

In python, the variables are defined with the two types

of scopes.

Global variables

Local variables

The variable defined outside any function is known to

have a global scope, whereas the variable defined

inside a function is known to have a local scope.

#---------------------------------------------------------------------

Local Variable

def print\_message():

message = "hello !! I am going to print a message." # the variable message is local to the function itself

print(message)

print\_message()

print(message) # this will cause an error since a local

variable cannot be accessible here.

#--------------------------------------------

Global Variable

def calculate(\*args):

sum=0

for arg in args:

sum = sum +arg

print("The sum is",sum)

sum=0

calculate(10,20,30) #60 will be printed as the sum

print("Value of sum outside the function:",sum) # 0 will be printed Output:

#---------------------------------------------------------------------

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#----------------------------------------

Python Built-in Functions

The Python built-in functions are defined as the

functions whose functionality is pre-defined in Python.

The python interpreter has several functions that are

always present for use. These functions

are known as Built-in Functions. There are several

built-in functions in Python

Python 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.

------------

# integer number

integer = -40

print('Absolute value of -40 is:', abs(integer))

output:

40

--------------

# floating number

floating = -40.83

print('Absolute value of -40.83 is:', abs(floating))

output:

Absolute value of -40.83 is: 40.83

#------------------------------------------------------

Python bin() Function

The python bin() function is used to return the binary

representation of a specified integer.

A result always starts with the prefix 0b.

Python bin() Function Example

-------------------------

x = 28

y = bin(x)

print (y)

output:

0b11100

#----------------------------------------------

Python sum() Function

As the name says, python sum() function is used to get

the sum of numbers of an iterable, i.e., list.

Python sum() Function Example

------------------------

s = sum([1, 2,4 ])

print(s)

s = sum([1, 2, 4], 10)

print(s)

output:

7

17

#----------------------------------------------------

Python float()

The python float() function returns a floating-point

number from a number or string.

Python float() Example

---------------

# for integers

print(float(9))

# for floats

print(float(8.19))

# for string floats

print(float("-24.27"))

# for string floats with whitespaces

print(float(" -17.19\n"))

# string float error

print(float("xyz"))

#-------------------------------------------

Python format() Function

The python format() function returns a formatted

representation of the given value.

--------------------

Python format() Function Example

# d, f and b are a type

# integer

print(format(123, "d"))

# float arguments

print(format(123.4567898, "f"))

# binary format

print(format(12, "b"))

output:

123

123.456790

1100

#----------------------------------------------------

Python iter() Function

The python iter() function is used to return an iterator

object. It creates an object which can be

iterated one element at a time.

Python iter() Function Example

--------------------

# list of numbers

list = [1,2,3,4,5]

listIter = iter(list)

# prints '1'

print(next(listIter))

# prints '2'

print(next(listIter))

# prints '3'

print(next(listIter))

# prints '4'

print(next(listIter))

# prints '5'

print(next(listIter))

#---------------------------------------------

Python len() Function

The python len() function is used to return the length

(the number of items) of an object.

Python len() Function Example

--------------------

strA = 'Python'

print(len(strA))

output:

6

#----------------------------------------------------------

Python list()

The python list() creates a list in python.

Python list() Example

------------

# empty list

print(list())

# string

String = 'abcde'

print(list(String))

# tuple

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

print(list(Tuple))

# list

List = [1,2,3,4,5]

print(list(List))

output:

[]

['a', 'b', 'c', 'd', 'e']

[1, 2, 3, 4, 5]

[1, 2, 3, 4, 5]

#---------------------------------------

Python object()

The python object() returns an empty object. It is a

base for all the classes and holds the built-in

properties and methods which are default for all the

classes.

----------------------

Python object() Example

obj = object()

print(type(obj))

print(dir(obj))

output:

<class 'object'>

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_']

#-------------------------------------------------

Python complex()

Python complex() function is used to convert numbers or

string into a complex number. This method takes two

optional parameters and returns a complex number. The

first parameter is called a real and second as imaginary

parts.

# Python complex() function example

# Calling function

a = complex(1) # Passing single parameter

b = complex(1,2) # Passing both parameters

# Displaying result

print(a)

print(b)

output:

(1+0j)

(1+2j)

#---------------------------------

Python dir() Function

Python dir() function returns the list of names in the

current local scope. If the object on which

method is called has a method named \_\_dir\_\_(), this

method will be called and must return the list of

attributes.

It takes a single object type argument.

Python dir() Function Example

-----------

# Calling function

att = dir()

# Displaying result

print(att)

output:

['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_cached\_\_',

'\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_',

'\_\_package\_\_', '\_\_spec\_\_']

#----------------------------------

Python divmod() Function

Python divmod() function is used to get remainder and

quotient of two numbers. This function takes two numeric

arguments and returns a tuple. Both arguments are

required and numeric

Python divmod() Function Example

------------------

# Python divmod() function example

# Calling function

result = divmod(10,2)

# Displaying result

print(result)

output:

5, 0)

#-----------------------

Python dict()

Python dict() function is a constructor which creates a

dictionary. Python dictionary provides three different

constructors to create a dictionary:

If no argument is passed, it creates an empty dictionary.

If a positional argument is given, a dictionary is

created with the same key-value pairs.

Otherwise, pass an iterable object.

If keyword arguments are given, the keyword arguments

and their values are added to the dictionary created

from the positional argument.

Python dict() Example

-----------------

# Calling function

result = dict() # returns an empty dictionary

result2 = dict(a=1,b=2)

# Displaying result

print(result)

print(result2)

output:

{}

{'a': 1, 'b': 2}

#----------------------------------

Python hash() Function

Python hash() function is used to get the hash value of

an object. Python calculates the hash value by using the

hash algorithm. The hash values are integers and used to

compare dictionary keys during a dictionary lookup. We

can hash only the types which are given below:

Hashable types: \* bool \* int \* long \* float \* string \* Unicode \* tuple \* code object.

Python hash() Function Example

--------------------------

# Calling function

result = hash(21) # integer value

result2 = hash(22.2) # decimal value

# Displaying result

print(result)

print(result2)

#-----------------------------------------------------------

Python min() Function

Python min() function is used to get the smallest element

from the collection. This function takes two arguments,

first is a collection of elements and second is key, and

returns the smallest element from the collection.

Python min() Function Example

----------------------------

# Calling function

small = min(2225,325,2025) # returns smallest element

small2 = min(1000.25,2025.35,5625.36,10052.50)

# Displaying result

print(small)

print(small2)

#-------------------------------------------

Python set() Function

In python, a set is a built-in class, and this function

is a constructor of this class. It is used to create a

new set using elements passed during the call. It takes

an iterable object as an argument and returns a new set

object.

Python set() Function Example

-------------

# Calling function

result = set() # empty set

result2 = set('12')

result3 = set('datascience')

# Displaying result

print(result)

print(result2)

print(result3)

output:

set()

{'2', '1'}

{'i', 'c', 'a', 't', 'n', 's', 'd', 'e'}

#------------------------------------------------

Python hex() Function

Python hex() function is used to generate hex value of

an integer argument. It takes an integer argument

and returns an integer converted into a hexadecimal

string. In case, we want to get a hexadecimal value

of a float, then use float.hex() function.

Python hex() Function Example

--------------

# Calling function

result = hex(1)

# integer value

result2 = hex(342)

# Displaying result

print(result)

print(result2)

output:

0x1

0x156

#--------------------------------------

Python id() Function

Python id() function returns the identity of an object.

This is an integer which is guaranteed to be unique.

This function takes an argument as an object and returns

a unique integer number which represents identity.

Two objects with non-overlapping lifetimes may have the

same id() value.

Python id() Function Example

# Calling function

val = id("python") # string object

val2 = id(1200) # integer object

val3 = id([25,336,95,236,92,3225]) # List object

# Displaying result

print(val)

print(val2)

print(val3)

#------------------------------------

Python sorted() Function

Python sorted() function is used to sort elements. By

default, it sorts elements in an ascending order

but can be sorted in descending also. It takes four

arguments and returns a collection in sorted order.

In the case of a dictionary, it sorts only keys, not

values.

Python sorted() Function Example

---------------

str = "pythonpoint" # declaring string

# Calling function

sorted1 = sorted(str) # sorting string

# Displaying result

print(sorted1)

output:

['h', 'i', 'n', 'n', 'o', 'o', 'p', 'p', 't', 't', 'y']

#--------------------------------------------------

Python next() Function

Python next() function is used to fetch next item from

the collection. It takes two arguments, i.e.,

an iterator and a default value, and returns an element.

This method calls on iterator and throws an error if no

item is present. To avoid the error, we can set a

default value.

Python next() Function Example

-------------

number = iter([256, 32, 82]) # Creating iterator

# Calling function

item = next(number)

# Displaying result

print(item)

# second item

item = next(number)

print(item)

# third item

item = next(number)

print(item)

#------------------------------------

Python input() Function

Python input() function is used to get an input from the

user. It prompts for the user input and reads a line.

After reading data, it converts it into a string and

returns it. It throws an error EOFError if EOF is read.

Python input() Function Example

----------------

# Calling function

val = input("Enter a value: ")

# Displaying result

print("You entered:",val)

#--------------------------------------------------

Python int() Function

Python int() function is used to get an integer value.

It returns an expression converted into an integer

number. If the argument is a floating-point, the

conversion truncates the number. If the argument is

outside the integer range, then it converts the number

into a long type.

If the number is not a number or if a base is given,

the number must be a string.

Python int() Function Example

-----------------

# Calling function

val = int(10) # integer value

val2 = int(10.52) # float value

val3 = int('10') # string value

# Displaying result

print("integer values :",val, val2, val3)

#-------------------------------------------------

Python oct() Function

Python oct() function is used to get an octal value of

an integer number. This method takes an argument

and returns an integer converted into an octal string.

It throws an error TypeError,

if argument type is other than an integer.

Python oct() function Example

--------------------

# Calling function

val = oct(10)

# Displaying result

print("Octal value of 10:",val)

output:

Octal value of 10: 0o12

#----------------------------------

Python pow() Function

The python pow() function is used to compute the power

of a number. It returns x to the power of y.

If the third argument(z) is given, it returns x to the

power of y modulus z

Python pow() function Example

---------------------

# positive x, positive y (x\*\*y)

print(pow(4, 2))

# negative x, positive y

print(pow(-4, 2))

# positive x, negative y (x\*\*-y)

print(pow(4, -2))

# negative x, negative y

print(pow(-4, -2))

#--------------------------------------------

Python print() Function

The python print() function prints the given object

to the screen or other standard output devices.

Python print() function Example

--------------------

print("Python is programming language.")

x = 7

# Two objects passed

print("x =", x)

y = x

# Three objects passed

print('x =', x, '= y')

#------------------------------

Python range() Function

The python range() function returns an immutable sequence

of numbers starting from 0 by default, increments by

1 (by default) and ends at a specified number.

Python range() function Example

-----------------

# empty range

print(list(range(0)))

# using the range(stop)

print(list(range(4)))

# using the range(start, stop)

print(list(range(1,7 )))

#------------------------------------------------

Python reversed() Function

The python reversed() function returns the reversed

iterator of the given sequence.

Python reversed() function Example

# for string

String = 'Java'

print(list(reversed(String)))

# for tuple

Tuple = ('J', 'a', 'v', 'a')

print(list(reversed(Tuple)))

# for range

Range = range(8, 12)

print(list(reversed(Range)))

# for list

List = [1, 2, 7, 5]

print(list(reversed(List)))

#--------------------------------------

Python round() Function

The python round() function rounds off the digits of a

number and returns the floating point number.

Python round() Function Example

# for integers

print(round(10))

# for floating point

print(round(10.8))

# even choice

print(round(6.6))

output:

10

11

7

#--------------------------------------------

Python str

The python str() converts a specified value into a string.

Python str() Function Example

s1="hai"

s=s1 +","+str(25)

print(s)

output:

hai,25

#------------------------------------------

Python tuple() Function

The python tuple() function is used to create a tuple

object.

Python tuple() Function Example

--------------

t1 = tuple()

print('t1=', t1)

# creating a tuple from a list

t2 = tuple([1, 6, 9])

print('t2=', t2)

# creating a tuple from a string

t1 = tuple('Java')

print('t1=',t1)

# creating a tuple from a dictionary

t1 = tuple({4: 'four', 5: 'five'})

print('t1=',t1)

#---------------------------------------------

Python type()

The python type() returns the type of the specified

object if a single argument is passed to the type()

built in function. If three arguments are passed,

then it returns a new type object.

Python type() Function Example

-------------------------

List = [4, 5]

print(type(List))

Dict = {4: 'four', 5: 'five'}

print(type(Dict))

#-----------------------------------------

Python Lambda Functions

Python Lambda function is known as the anonymous function

that is defined without a name. Python allows us to not

declare the function in the standard manner, i.e., by

using the def keyword. Rather, the anonymous functions

are declared by using the lambda keyword. However,

Lambda functions can accept any number of arguments,

but they can return only one value in the form of

expression.

Syntax

lambda arguments : expression

The expression is executed and the result is returned:

Add 10 to argument a, and return the result:

x = lambda a : a + 10

print(x(5))

output:

15

#----------------------------------------------------------------

Lambda functions can take any number of arguments:

Multiply argument a with argument b and return the result:

x = lambda a, b : a \* b

print(x(5, 6))

output:

30

#------------------------------------------------

Summarize argument a, b, and c and return the result:

x = lambda a, b, c : a + b + c

print(x(5, 6, 2))

output:

13

#--------------------------------------------------

Why Use Lambda Functions?

The power of lambda is better shown when you use them as

an anonymous function inside another function.

Say you have a function definition that takes one argument,

and that argument will be multiplied with an unknown number:

def myfunc(n):

return lambda a : a \* n

#------------------------------------------------------------------

Use that function definition to make a function that always

doubles the number you send in:

Example

------------

def myfunc(n):

return lambda a : a \* n

mydoubler = myfunc(2)

print(mydoubler(11))

output:

22

#---------------------------------------------------------

Or, use the same function definition to make a function

that always triples the number you send in:

def myfunc(n):

return lambda a : a \* n

mytripler = myfunc(3)

print(mytripler(11))

output:

33

#--------------------------------------------------

or, use the same function definition to make both functions, in the same program:

def myfunc(n):

return lambda a : a \* n

mydoubler = myfunc(2)

mytripler = myfunc(3)

print(mydoubler(11))

print(mytripler(11))

#------------------------------------------------

Python Modules

A python module can be defined as a python program file

which contains a python code including python functions,

class, or variables. In other words, we can say that our

python code file saved with the extension (.py) is treated

as the module. We may have a runnable code inside the python

module.

Modules in Python provides us the flexibility to organize

the code in a logical way.

To use the functionality of one module into another, we must

have to import the specific module.

#------------------------------------

Let's create the module named as file.py.

--------------------

file.py

#displayMsg prints a message to the name being passed.

def displayMsg(name):

print("Hi "+name)

def show():

print("show here")

#---------------------------------

Loading the module in our python code

We need to load the module in our python code to use its functionality. Python provides two

types of statements as defined below.

1, The import statement

2, The from-import statement

The import statement

The import statement is used to import all the

functionality of one module into another. Here,

we must notice that we can use the functionality

of any python source file by importing that

file as the module into another python source file.

We can import multiple modules with a single import

statement, but a module is loaded once regardless

of the number of times, it has been imported into

our file.

The syntax to use the import statement is given below.

import module1,module2,........ module n

#----------------------------

Hence, if we need to call the function displayMsg() defined in the file file.py, we have to

import that file as a module into our module as shown in the example below.

---------------

import file

name = input("Enter the name:")

file.displayMsg(name)

file.show()

output:

Enter the name:kochi

Hi kochi

show here

#---------------------------------

The from-import statement

Instead of importing the whole module into the namespace, python provides the flexibility to import only the

specific attributes of a module. This can be done by using from? import statement. The syntax to use the from-import

statement is given below.

from < module-name> import <name 1>, <name 2>..,<name n>

-----------------------------

Consider the following module named as calculation which contains three functions as summation, multiplication,

and divide.

calculation.py

-----------------

#place the code in the calculation.py

def summation(a,b):

return a+b

def multiplication(a,b):

return a\*b;

def divide(a,b):

return a/b;

#-----------------------------------------

Main.py:

from calculation import summation

#it will import only the summation() from calculation.py

a = int(input("Enter the first number"))

b = int(input("Enter the second number"))

print("Sum = ",summation(a,b)) #we do not need to specify the module name while accessing summation()

output:

Enter the first number10

Enter the second number20

Sum = 30

#-----------------------------------------------------

The from...import statement is always better to use if we know the attributes to be imported from the module

in advance. It doesn't let our code to be heavier. We can also import all the attributes from a module by using \*.

Consider the following syntax.

from <module> import \*

Example

from calculation import \*

#-----------------------

Python Datetime

Python Dates

A date in Python is not a data type of its own, but

we can import a module

named datetime to work with dates as

date objects.

Import the datetime module and display the current date:

#--------------------------------------------------------

import datetime

x = datetime.datetime.now()

print(x)

output:

2022-09-17 15:54:58.143493

Date Output

The date contains year, month, day, hour, minute, second, and microsecond.

---------------------

The datetime module has many methods to return information about the date object.

Return the year and name of weekday:

----------

Creating Date Objects

To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

-------------------

import datetime

x = datetime.datetime(2020, 5, 17)

print(x)

output:

2020-05-17 00:00:00

----------------------

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone),

but they are optional, and has a default value of 0

------------------------------

The strftime() Method

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

legal format codes:

---------------------------

%a Weekday, short version

-------------------------------------

import datetime

x = datetime.datetime.now()

print(x.strftime("%a"))

output:

Fri

--------

%A Weekday, full version

import datetime

x = datetime.datetime.now()

print(x.strftime("%A"))

output:

Friday

-------------------------

%w Weekday as a number 0-6, 0 is Sunday

import datetime

x = datetime.datetime.now()

print(x.strftime("%w"))

output:

5

----------------

%d Day of month 01-31 27

---------------------

%b Month name, short version Jan

----------------------

%B Month name, full version January

-------------------------

%m Month as a number 01-12 1

-------------------------

%y Year, short version, without century 23

-----------------------------------

%Y Year, full version 2023

----------------------------

%H Hour 00-23 10

-------------------------

%I Hour 00-12 05

----------------------------

%p AM/PM PM

-------------------

%M Minute 00-59 41

-------------------------

%S Second 00-59 08

-----------------------

%f Microsecond 000000-999999 548513

--------------------

%j Day number of year 001-366 365

------------------------------

%W Week number of year, Monday as the first day of week, 00-53 52

---------------------

%C Ce

--------------

Python Math Module

Python math module is defined as the most famous

mathematical functions,

which includes trigonometric functions,

representation functions, logarithmic functions, etc.

math.pow(x,y)

This method returns the power of the x corresponding

to the value of y.

If value of x is negative or y is not

integer value than it raises a ValueError.

-------------------

import math

number = math.pow(10,2)

print("The power of number:",number)

output:

The power of number: 100.0

-------------------

math.floor(x)

This method C It returns the less than or equal value

to x.

--------------

import math

number = math.floor(10.25201)

print("The floor value is:",number)

output:

The floor value is: 10

-----------------------------

math.ceil(x)

This method returns the ceil value of the x. It returns

the greater than or equal value to x.

----------------

import math

number = math.ceil(10.25201)

print("The ceil value is:",number)

output:

The ceil value is: 11

--------------------------------

math.factorial()

This method returns the factorial of the given

number x. If x is not integral,

it raises a ValueError.

-------------

import math

number = math.factorial(7)

print("The factorial of number:",number)

output:

The factorial of number: 5040

-------------------------

math.modf(x)

This method returns the fractional and integer parts

of x. It carries the sign of x is float.

import math

number = math.modf(44.5)

print("The modf of number:",number)

output:

The modf of number: (0.5, 44.0)

-------------------------------

Python has also a built-in module called math, which

extends the list of mathematical functions.

To use it, you must import the math module:

------------------

import math

x = math.sqrt(64)

print(x)

output:

8

----------------------

The math.pi constant, returns the value of PI

(3.14...):

---------------------------

import math

x = math.pi

print(x)

-------------------

Python math.fmod() Method

Return the remainder of x/y:

--------------------

# Import math Library

import math

# Return the remainder of x/y

print(math.fmod(20, 4))

print(math.fmod(20, 3))

print(math.fmod(15, 6))

print(math.fmod(-10, 3))

#print(math.fmod(0, 0)) Error, div with zero

output:

0.0

2.0

3.0

-1.0

---------------

Python math.fsum() Method

Return the sum of all items:

-----------------------

# Import math Library

import math

# Print the sum of all items

print(math.fsum([1, 2, 3, 4, 5]))

print(math.fsum([100, 400, 340, 500]))

print(math.fsum([1.7, 0.3, 1.5, 4.5]))

output:

15.0

1340.0

8.0

----------------------

Python math.gcd() Method

Find the greatest common divisor of the two integers:

-----------------------------

#Import math Library

import math

#find the the greatest common divisor of the two

integers

print (math.gcd(3, 6))

print (math.gcd(6, 12))

print (math.gcd(12, 36))

print (math.gcd(-12, -36))

print (math.gcd(5, 12))

print (math.gcd(10, 0))

print (math.gcd(0, 34))

print (math.gcd(0, 0))

output:

3

6

12

12

1

10

34

0

-----------------------

Python math.isinf() Method

Check whether a value is infinity or not:

-----------------------------

# Import math Library

import math

# Check whether the values are infinite or not

print(math.isinf(56))

print(math.isinf(-45.34))

print(math.isinf(+45.34))

print(math.isinf(math.inf))

print(math.isinf(float("nan")))

print(math.isinf(float("inf")))

print(math.isinf(float("-inf")))

print(math.isinf(-math.inf))

output:

False

False

False

True

False

True

True

True

----------------------------------

Python math.isnan() Method

Check whether a value is NaN or not:

-------------------

# Import math Library

import math

# Check whether some values are NaN or not

print (math.isnan (56))

print (math.isnan (-45.34))

print (math.isnan (+45.34))

print (math.isnan (math.inf))

print (math.isnan (float("nan")))

print (math.isnan (float("inf")))

print (math.isnan (float("-inf")))

print (math.isnan (math.nan))

output:

False

False

False

False

True

False

False

True

---------------------

Python Random Module

Python Random module is an in-built module of Python

which is used to generate random numbers. These are

pseudo-random numbers means these are not truly

random. This module can be used to perform random

actions such as generating random numbers, print

random a value for a list or string, etc.

Printing a random value from a list

--------------

# import random

import random

# prints a random value from the list

list1 = [1, 2, 3, 4, 5, 6]

print(random.choice(list1))

output:

2

-----------------------------

Creating Random Integers

random.randint() method is used to generate random

integers between the given range.

Syntax :

randint(start, end)

# Python3 program explaining work

# of randint() function

---------------------------------

# import random module

import random

# Generates a random number between

# a given positive range

r1 = random.randint(5, 15)

print("Random number between 5 and 15 is %d" % (r1))

# Generates a random number between

# two given negative range

r2 = random.randint(-10, -2)

print("Random number between -10 and -2 is %d" % (r2))

output:

Random number between 5 and 15 is 13

Random number between -10 and -2 is -8

-------------------------------------------------

Creating Random Floats

random.random() method is used to generate random

floats between 0.0 to 1.0

Syntax:

random.random()

# Python3 program to demonstrate

# the use of random() function .

--------------------------------

# import random

from random import random

# Prints random item

print(random())

output:

0.4981855042679951

----------------------

Shuffling List

random.shuffle() method is used to shuffle a sequence

(list). Shuffling means changing the position of the

elements of the sequence. Here, the shuffling

operation is inplace.

Syntax:

random.shuffle(sequence)

--------------------------------

# import the random module

import random

# declare a list

sample\_list = [1, 2, 3, 4, 5]

print("Original list : ")

print(sample\_list)

# first shuffle

random.shuffle(sample\_list)

print("\nAfter the first shuffle : ")

print(sample\_list)

# second shuffle

random.shuffle(sample\_list)

print("\nAfter the second shuffle : ")

print(sample\_list)

---------------------------------------

Python statistics module

Python statistics module provides the functions to

mathematical statistics of numeric data.

There are some popular statistical functions defined

in this module.

mean() function

The mean() function is used to calculate the arithmetic

mean of the numbers in the list.

-------------------------------

import statistics

# list of positive integer numbers

datasets = [5, 2, 7, 4, 2, 6, 8]

x = statistics.mean(datasets)

# Printing the mean

print("Mean is :", x)

-----------------------------

median() function

The median() function is used to return the middle

value of the numeric data in the list.

-----------------------

import statistics

datasets = [4, -5, 6, 6, 9, 4, 5, -2]

# Printing median of the

# random data-set

print("Median of data-set is : % s " % (statistics.median(datasets)))

---------------------

mode() function

The mode() function returns the most common data that

occurs in the list.

-------------------------------

import statistics

# declaring a simple data-set consisting of real valued

positive integers.

dataset =[2, 4, 7, 7, 2, 2, 3, 6, 6, 8]

# Printing out the mode of given data-set

print("Calculated Mode % s" % (statistics.mode(dataset)))

-----------------------

stdev() function

The stdev() function is used to calculate the standard

deviation on a given sample which is

available in the form of the list.

----------------------------------

import statistics

# creating a simple data - set

sample = [7, 8, 9, 10, 11]

# Prints standard deviation

print("Standard Deviation of sample is % s " % (statistics.stdev(sample)))

-----------------------------------

median\_low()

The median\_low function is used to return the low median of numeric data in the list.

-------------------

import statistics

# simple list of a set of integers

set1 = [4, 6, 2, 5, 7, 7]

# Note: low median will always be a member of the data-set.

# Print low median of the data-set

print("Low median of data-set is % s " % (statistics.median\_low(set1)))

----------------------------

median\_high()

The median\_high function is used to return the high median of

numeric data in the list.

import statistics

# list of set of the integers

dataset = [2, 1, 7, 6, 1, 9]

print("High median of data-set is %s " % (statistics.median\_high(dataset)))

-----------------------

Python JSON

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation

JSON in Python

Python has a built-in package called json, which can be

used to work with

JSON data.

Import the json module:

import json

Parse JSON - Convert from JSON to Python

If you have a JSON string, you can parse it by using the json.loads() method.

Convert from JSON to Python:

-----------------

import json

# some JSON:

x = '{ "name":"John", "age":30, "city":"New York"}'

# parse x:

y = json.loads(x)

print(y)

# the result is a Python dictionary:

print(y["age"])

output:

{'name': 'John', 'age': 30, 'city': 'New York'}

30

-----------------------------

Convert from Python to JSON

If you have a Python object, you can convert it into a JSON string by using

the json.dumps() method.

Convert from Python to JSON:

-------------------

import json

# a Python object (dict):

x = {

"name": "John",

"age": 30,

"city": "New York"

}

# convert into JSON:

y = json.dumps(x)

# the result is a JSON string:

print(y)

output:

{"name": "John", "age": 30, "city": "New York"}

-------------------------------

You can convert Python objects of the following types,

into JSON strings:

dict

list

tuple

string

int

float

True

False

None

Convert Python objects into JSON strings, and print the

values:

-------------------------

import json

print(json.dumps({"name": "John", "age": 30}))

print(json.dumps(["apple", "bananas"]))

print(json.dumps(("apple", "bananas")))

print(json.dumps("hello"))

print(json.dumps(42))

print(json.dumps(31.76))

print(json.dumps(True))

print(json.dumps(False))

output:

{"name": "John", "age": 30}

["apple", "bananas"]

["apple", "bananas"]

"hello"

42

31.76

true

false

-----------------

Convert a Python object containing all the legal data

types:

----------------

import json

x = {

"name": "John",

"age": 30,

"married": True,

"divorced": False,

"children": ("Ann","Billy"),

"pets": None,

"cars": [

{"model": "BMW 230", "mpg": 27.5},

{"model": "Ford Edge", "mpg": 24.1}

]

}

print(json.dumps(x))

output:

{"name": "John", "age": 30, "married": true, "divorced": false, "children": ["Ann", "Billy"], "pets": null, "cars": [{"model": "BMW 230", "mpg": 27.5}, {"model": "Ford Edge", "mpg": 24.1}]}

---------------------------

Format the Result

The example above prints a JSON string, but it is not

very easy to read, with no indentations

and line breaks.

The json.dumps() method has parameters to make it

easier to read the result:

Use the indent parameter to define the numbers of

indents:

---------------

import json

x = {

"name": "John",

"age": 30,

"married": True,

"divorced": False,

"children": ("Ann","Billy"),

"pets": None,

"cars": [

{"model": "BMW 230", "mpg": 27.5},

{"model": "Ford Edge", "mpg": 24.1}

]

}

# use four indents to make it easier to read the result:

print(json.dumps(x, indent=4))

output:

{

"name": "John",

"age": 30,

"married": true,

"divorced": false,

"children": [

"Ann",

"Billy"

],

"pets": null,

"cars": [

{

"model": "BMW 230",

"mpg": 27.5

},

{

"model": "Ford Edge",

"mpg": 24.1

}

]

}

------------------

You can also define the separators, default value is (", " , ": "), which means

using a comma and a space

to separate each object, and a colon and a space to separate keys from values:

Use the separators parameter to change the default separator:

---------------------------

import json

x = {

"name": "John",

"age": 30,

"married": True,

"divorced": False,

"children": ("Ann","Billy"),

"pets": None,

"cars": [

{"model": "BMW 230", "mpg": 27.5},

{"model": "Ford Edge", "mpg": 24.1}

]

}

# use . and a space to separate objects, and a space, a = and a space to separate keys from their values:

print(json.dumps(x, indent=4, separators=(". ", " = ")))

output:

{

"name" = "John".

"age" = 30.

"married" = true.

"divorced" = false.

"children" = [

"Ann".

"Billy"

].

"pets" = null.

"cars" = [

{

"model" = "BMW 230".

"mpg" = 27.5

}.

{

"model" = "Ford Edge".

"mpg" = 24.1

}

]

}

-----------------------

Order the Result

The json.dumps() method has parameters to order the keys in the result:

Use the sort\_keys parameter to specify if the result should be sorted or not:

--------------------------

import json

x = {

"name": "John",

"age": 30,

"married": True,

"divorced": False,

"children": ("Ann","Billy"),

"pets": None,

"cars": [

{"model": "BMW 230", "mpg": 27.5},

{"model": "Ford Edge", "mpg": 24.1}

]

}

# sort the result alphabetically by keys:

print(json.dumps(x, indent=4, sort\_keys=True))

output:

{

"age": 30,

"cars": [

{

"model": "BMW 230",

"mpg": 27.5

},

{

"model": "Ford Edge",

"mpg": 24.1

}

],

"children": [

"Ann",

"Billy"

],

"divorced": false,

"married": true,

"name": "John",

"pets": null

}

-------------------------------

Python Exception

An exception can be defined as an unusual

condition in a program resulting in the

interruption in the flow of the program.

Whenever an exception occurs, the program stops the

execution, and thus the further code is not executed.

Therefore, an exception is the run-time errors that

are unable to handle to Python script. An exception

is a Python object that represents an error

Python provides a way to handle the exception so

that the code can be executed without any

interruption. If we do not handle the exception,

the interpreter doesn't execute all the code

that exists after the exception.

Python has many built-in exceptions that enable

our program to run without interruption and give

the output. These exceptions are given below:

Common Exceptions:

Python provides the number of built-in exceptions,

but here we are describing the common standard

exceptions. A list of common exceptions that can

be thrown from a standard Python program

is given below.

1. ZeroDivisionError: Occurs when a number is divided by zero.

2. NameError: It occurs when a name is not found. It may be

local or global.

3. IndentationError: If incorrect indentation is given.

4. IOError: It occurs when Input Output operation fails.

5. EOFError: It occurs when the end of the file is reached,

and yet operations are being performed.

The problem without handling exceptions

Exception is an abnormal condition that halts the

execution of the program.

Suppose we have two variables a and b, which take

the input from the user and perform the division

of these values. What if the user entered the zero

as the denominator? It will interrupt the program

execution and through a ZeroDivision exception.

#------------------------------

#Example

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b

print("a/b = %d" % c)

#other code:

print("Hi I am other part of the program")

output:

Enter a:5

Enter b:0

Traceback (most recent call last):

File "F:\DATASCIENCE\PROJECTS\MLProjects\TestWorks.py", line 4, in <module>

c = a/b

ZeroDivisionError: division by zero

#--------------------------------

The above program is syntactically correct, but it

through the error because of unusual input. That

kind of programming may not be suitable or

recommended for the projects because these

projects are required uninterrupted execution.

That's why an exception-handling plays an essential

role in handling these unexpected exceptions.

-------------------------------------------------------------------------------

The try-except statement:

If the Python program contains suspicious code that may throw

the exception, we must place that code in the try block. The

try block must be followed with the except statement,

which contains a block of code that will be executed

if there is some exception in the try block.

Syntax

try:

#block of code

except Exception1:

#block of code

except Exception2:

#block of code

#other code

-----------------

#Example

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b

print("result:",c)

print("hi..")

except:

print("Can't divide with zero")

print("outside exception")

output:

Enter a:5

Enter b:0

Can't divide with zero

outside exception

--------------------------------

We can also use the else statement with the try-except statement

in which, we can place the code which will be executed in the

scenario if no exception occurs in the try block.

The syntax to use the else statement with the try-except

statement is given below.

try:

#block of code

except Exception1:

#block of code

except Exception2:

#block of code

...

else:

#this code executes if no except block is executed

--------------------

#Example

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b

print("a/b = %d" % c)

# Using Exception with except statement. If we print(Exception) it will return exception class

except Exception:

print("can't divide by zero")

print(Exception)

else:

print("Hi I am else block")

print("outside exception")

output:

Enter a:5

Enter b:0

can't divide by zero

<class 'Exception'>

outside exception

------------------------------------

The except statement with no exception

Python provides the flexibility not to specify the name of exception

with the exception statement.

-----------------

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b;

print("a/b = %d"%c)

except:

print("can't divide by zero")

else:

print("Hi I am else block")

print("outside exception")

output:

Enter a:5

Enter b:0

can't divide by zero

outside exception

------------------------------

The except statement using with exception variable

We can use the exception variable with the except statement. It is used by using the as

keyword. this object will return the cause of the exception.

------------------------------

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b

print("a/b = %d"%c)

# Using exception object with the except statement

except Exception as e:

print("can't divide by zero")

print(e)

else:

print("Hi I am else block")

print("outside exception")

output:

Enter b:0

can't divide by zero

division by zero

outside exception

------------------

Points to remember

1. Python facilitates us to not specify the exception with the except statement.

2. We can declare multiple exceptions in the except statement since the try block

may contain the statements which throw the different type of exceptions.

3. We can also specify an else block along with the try-except statement,

which will be executed if no exception is raised in the try block.

4. The statements that don't throw the exception should be placed inside the else block.

------------------------------

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b

print("a/b = %d"%c)

# multiple exception

except IOError as e1:

print("IO Exception",e1)

except ArithmeticError as e2:

print("Arithmetic Exception", e2)

else:

print("Successfully Done")

print("outside exception")

output:

Enter a:5

Enter b:0

Arithmetic Exception division by zero

outside exception

---------------------------------------------------

Declaring Multiple Exceptions

The Python allows us to declare the multiple exceptions with the

except clause. Declaring multiple exceptions is useful in the

cases where a try block throws multiple exceptions.

The syntax is given below.

Syntax

try:

#block of code

except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>)

#block of code

else:

#block of code

-----------------------

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b

print("a/b = %d"%c)

# multiple exception

except(ArithmeticError, IOError):

print("Arithmetic Exception")

else:

print("Successfully Done")

print("outside exception")

output:

Enter a:5

Enter b:0

Arithmetic Exception

outside exception

-----------------------------------

The try...finally block

Python provides the optional finally statement, which is used with the try statement.

It is executed no matter what exception occurs and used to release the external resource.

The finally block provides a guarantee of the execution.

We can use the finally block with the try block in which we can place the necessary code,

which must be executed before the try statement throws an exception.

The syntax to use the finally block is given below.

Syntax

try:

# block of code

# this may throw an exception

finally:

# block of code

# this will always be executed

-----------------------------

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

c = a/b

print("a/b = %d"%c)

# Using exception object with the except statement

except Exception as e:

print("can't divide by zero")

print(e)

else:

print("Hi I am else block")

finally:

print("inside finally")

print("outside exception")

output:

Enter a:5

Enter b:0

can't divide by zero

division by zero

inside finally

outside exception

------------------------------------

Raising exceptions:

An exception can be raised forcefully by using the

raise clause in Python. It is useful in in that

scenario where we need to raise an exception to

stop the execution of the program.

The syntax to use the raise statement is given

below.

Syntax

raise Exception\_class(<value>)

Points to remember

1. To raise an exception, the raise statement is used. The exception class name follows it.

2. An exception can be provided with a value that can be given in the parenthesis.

3. To access the value "as" keyword is used. "e" is used as a reference variable which stores the value of the exception.

4. We can pass the value to an exception to specify the exception type.

-----------------------------

try:

age = int(input("Enter the age:"))

if(age<18):

raise ValueError

else:

print("the age is valid")

except Exception:

print("The age is not valid")

print("outside exception")

output:

Enter the age:12

The age is not valid

outside exception

-------------------------

# Raise the exception with message

try:

num = int(input("Enter a positive integer: "))

if(num <= 0):

# we can pass the message in the raise statement

raise ValueError("That is a negative number!")

except Exception as e:

print(e)

output:

Enter a positive integer: -2

That is a negative number!

-----------------------------

try:

a = int(input("Enter a:"))

b = int(input("Enter b:"))

if b == 0:

raise ArithmeticError("Div with zero...")

else:

print("a/b = ",a/b)

except ArithmeticError as e:

print("The value of b can't be 0")

print("Error:",e)

output:

Enter a:10

Enter b:0

The value of b can't be 0

Error: Div with zero...

------------------------------------------------------

Python File Handling

Till now, we were taking the input from the console and

writing it back to the console to interact with the user.

Sometimes, it is not enough to only display the data on

the console. The data to be displayed may be very large,

and only a limited amount of data can be displayed on

the console since the memory is volatile, it is impossible

to recover the programmatically generated data again

and again.

The file handling plays an important role when the data

needs to be stored permanently into the file. A file is

a named location on disk to store related information.

We can access the stored information (non-volatile) after

the program termination. The file-handling implementation

is slightly lengthy or complicated in the other programming

language, but it is easier and shorter in Python.

In Python, files are treated in two modes as text or

binary. The file may be in the text or binary format,

and each line of a file is ended with the special

character.

Hence, a file operation can be done in the following order.

Open a file

Read or write - Performing operation

Close the file

Opening a file

Python provides an open() function that accepts two

arguments, file name and access mode in which the file

is accessed. The function returns a file object which

can be used to perform various operations like reading,

writing, etc.

Syntax:

file\_object = open(<file-name>, <access-mode>)

The files can be accessed using various modes like read,

write, or append. The following are the details about

the access mode to open a file.

r- It opens the file to read-only mode. The file pointer

exists at the beginning. The file is by default open in

this mode if no access mode is passed.

----------

rb- It opens the file to read-only in binary format.

The file pointer exists at the beginning of the file.

----------

r+ -It opens the file to read and write both. The file

pointer exists at the beginning of the file.

----------

rb+ -It opens the file to read and write both in binary

format. The file pointer exists at the beginning of the

file.

-----------

w -It opens the file to write only. It overwrites the

file if previously exists or creates a new one if no file

exists with the same name. The file pointer exists at

the beginning of the file.

-----------

wb -It opens the file to write only in binary format.

It overwrites the file if it exists previously or creates

a new one if no file exists. The file pointer exists at

the beginning of the file.

-----------

w+ -It opens the file to write and read both. It is

different from r+ in the sense that it overwrites the

previous file if one exists whereas r+ doesn't overwrite

the previously written file. It creates a new file if

no file exists. The file pointer exists at the beginning

of the file.

----------------

wb+ -It opens the file to write and read both in binary

format. The file pointer exists at the beginning of the

file.

---------------

a -It opens the file in the append mode. The file

pointer exists at the end of the previously written

file if exists any. It creates a new file if no file

exists with the same name.

---------------

ab -It opens the file in the append mode in binary

format. The pointer exists at the end of the previously

written file. It creates a new file in binary format

if no file exists with the same name.

----------------

a+ -It opens a file to append and read both. The file

pointer remains at the end of the file if a file exists.

It creates a new file if no file exists with the same name.

---------------

ab+ -It opens a file to append and read both in binary

format. The file pointer remains at the end of the file.

---------------

Let's look at the simple example to open a file named

"sample.txt"

(stored in the same directory) in read mode and printing its content

on the console.

---------------

#opens the file file.txt in read mode

#fileptr = open("C:\\Users\\BAIJU\\Desktop\\sample.txt","r")

fileptr = open("C:/Users/BAIJU/Desktop/sample.txt","r")

if fileptr:

print("file is opened successfully")

else:

print("file not found")

----------------------------------------

In the above code, we have passed filename as a first

argument and opened file in read mode as we mentioned

r as the second argument. The fileptr holds the file

object and if the file is opened successfully, it will

execute the print statement

-------------------

The close() method

Once all the operations are done on the file, we must

close it through our Python script using the close()

method. Any unwritten information gets destroyed once

the close() method is called on a file object.

We can perform any operation on the file externally using

the file system which is the currently opened in Python;

hence it is good practice to close the file once all the

operations are done.

The syntax to use the close() method is given below.

Syntax

fileobject.close()

--------------

#opens and close the file file.txt in read mode

fileptr = open("D\\DATASET\\sample.txt","r")

if fileptr:

print("file is opened successfully")

#closes the opened file

fileptr.close()

----------------------

After closing the file, we cannot perform any operation

in the file. The file needs to be properly closed. If

any exception occurs while performing some operations

in the file then the program terminates

without closing the file.

--------------

The with statement

The with statement was introduced in python 2.5. The with statement

is useful in the case of manipulating the files. It is used in the

scenario where a pair of statements is to be executed with a block of

code in between.

The syntax to open a file using with the statement is given below.

with open(<file name>, <access mode>) as <file-pointer>:

#statement suite

-------------------

The advantage of using with statement is that it provides the

guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the with statement in the case of

files because, if the break, return, or exception occurs in the

nested block of code then it automatically closes the file, we don't

need to write the close() function. It doesn't let the file to corrupt.

-----------------

#with open("C:\\Users\\BAIJU\\Desktop\\sample.txt","r") as f:

with open("C:/Users/BAIJU/Desktop/sample.txt","r") as f:

content = f.read()

print(content)

-----------------------

Writing the 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 any file exists. The file pointer

is at the beginning of the file.

a: It will append the existing file. The file pointer is at the

end of the file. It creates a new file if no file exists.

------------------

# open the sample.txt in write mode. Create a new file if no such file exists.

fileptr = open("F\\DATASET\\sample.txt", "w")

# overwriting the content to the file

fileptr.write('''Python is the modern day language. It makes things so simple.

It is the fastest-growing programing language''')

# closing the opened the file

fileptr.close()

---------

We have opened the file in w mode. The sample.txt file doesn't exist,

it created a new file and we have written the content in the file using

the write() function.

------------

#open the sample.txt in append mode.

fileptr = open("sample.txt","a")

#appending the content of the file

fileptr.write(" Python has an easy syntax and user-friendly interaction.")

#closing the opened file

fileptr.close()

---------------

We can see that the content of the file is modified. We have opened the

file in a mode and it appended the content in the existing sample.txt.

------------------

To read a file using the Python script, the Python provides the read() method.

The read() method reads a string from the file. It can read the data in

the text as well as a binary format.

The syntax of the read() method is given below.

Syntax:

fileobj.read(<count>)

-----------------

Here, the count is the number of bytes to be read from the file starting

from the beginning of the file. If the count is not specified, then it may

read the content of the file until the end.

-----------------------

#open the file.txt in read mode. causes error if no such file exists.

fileptr = open("F:\\DATASET\\sample.txt","r")

#stores all the data of the file into the variable content

content = fileptr.read(10)

#prints the content of the file

print(content)

#closes the opened file

fileptr.close()

---------------------

In the above code, we have read the content of sample.txt by using

the read() function. We have passed count value as ten which means

it will read the first ten characters from the file.

If we use the following line, then it will print all content of the

file.

----------------------

fileptr = open("F:\\DATASET\\sample.txt","r")

content = fileptr.read()

print(content)

---------------------

Read file through for loop

We can read the file using for loop.

Example.

-------------------------------------------

#open the file.txt in read mode. causes an error if no such file exists.

fileptr = open("F:\\DATASET\\sample.txt","r");

#running a for loop

for i in fileptr:

print(i) # i contains each line of the file

-----------

Read Lines of the file

Python facilitates to read the file line by line by using a function

readline() method. 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.

Example which contains a function readline() that reads the first line

of our file "sample.txt" containing three lines.

----------------

#open the file.txt in read mode. causes error if no such file exists.

fileptr = open("C:\\Users\\BAIJU\\Desktop\\kochi.txt","r");

#stores all the data of the file into the variable content

content = fileptr.readline()

content1 = fileptr.readline()

#prints the content of the file

print(content)

print(content1)

#closes the opened file

fileptr.close()

------------------

We called the readline() function two times that's why it read two

lines from the file.

Python provides also the readlines() method which is used for the

reading lines. It returns the list of the lines till the end of

file(EOF) is reached.

----------------------------

#open the file.txt in read mode. causes error if no such file exists.

fileptr = open("C:\\Users\\BAIJU\\Desktop\\kochi.txt","r");

#stores all the data of the file into the variable content

content = fileptr.readlines()

#prints the content of the file

print(content)

#closes the opened file

fileptr.close()

---------------------

Creating a new file

The new file can be created by using one of the following access

modes with the function open().

x: it creates a new file with the specified name. It causes an error a

file exists with the same name.

a: It creates a new file with the specified name if no such file exists.

It appends the content

to the file if the file already exists with the specified name.

w: It creates a new file with the specified name if no such file exists.

It overwrites the existing file.

-----------------------

#open the file.txt in read mode. causes error if no such file exists.

fileptr = open("F:\\DATASET\\file2.txt","x")

print(fileptr)

if fileptr:

print("File created successfully")

---------------------

File Pointer positions

Python provides the tell() method which is used to print the byte number

at which the file pointer currently exists.

----------------------------

# open the file file2.txt in read mode

fileptr = open("F:\\DATASET\\file2.txt","r")

#initially the filepointer is at 0

print("The filepointer is at byte :",fileptr.tell())

#reading the content of the file

content = fileptr.read()

#after the read operation file pointer modifies. tell() returns the

location of the fileptr.

print("After reading, the filepointer is at:",fileptr.tell())

-----------------

Modifying file pointer position

In real-world applications, sometimes we need to change the file

pointer location externally since we may need to read or write the

content at various locations.

For this purpose, the Python provides us the seek() method which enables

us to modify the file pointer position externally.

The syntax to use the seek() method is given below.

Syntax:

<file-ptr>.seek(offset[, from)

---------------------

offset: It refers to the new position of the file pointer within the file.

from: It indicates the reference position from where the bytes are to be moved.

If it is set to 0, the beginning of the file is used as the reference position.

If it is set to 1, the current position of the file pointer is used as the

reference position. If it is set to 2, the end of the file pointer is used

as the reference position.

---------------

# open the file file2.txt in read mode

fileptr = open("F:\\DATASET\\file2.txt","r")

#initially the filepointer is at 0

print("The filepointer is at byte :",fileptr.tell())

#changing the file pointer location to 10.

fileptr.seek(10);

#tell() returns the location of the fileptr.

print("After reading, the filepointer is at:",fileptr.tell())

---------------

Python OS module

Renaming the file

The Python os module enables interaction with the operating system.

The os module provides the functions that are involved in file processing

operations like renaming, deleting, etc. It provides us the rename() method

to rename the specified file to a new name. The syntax to use the rename()

method is given below.

-------------

Syntax:

rename(current-name, new-name)

The first argument is the current file name and the second argument is

the modified name. We can change the file name bypassing these two arguments.

-----------------------

import os

#rename file2.txt to file3.txt

os.rename("C:\\Users\\BAIJU\\OneDrive\\Desktop\\sample.txt","C:\\Users\\BAIJU\\OneDrive\\Desktop\\kochi.txt")

print("renamed")

-----------------

Removing the file

The os module provides the remove() method which is used to remove

the specified file. The syntax to use the remove() method is given below.

syntax:

remove(file-name)

-----------------------

import os

#deleting the file named file3.txt

os.remove("file3.txt")

print("File removed")

------------------------

Creating the new directory

The mkdir() method is used to create the directories in the current

working directory. The syntax to create the new directory is given below.

Syntax:

mkdir(directory name)

-----------------------------

import os

#creating a new directory with the name new

os.mkdir("ktym")

print("folder created")

--------------------------

The getcwd() method

This method returns the current working directory.

The syntax to use the getcwd() method is given below.

Syntax

os.getcwd()

------------

import os

print(os.getcwd())

print("current directory")

---------------

Changing the current working directory

The chdir() method is used to change the current working directory to a

specified directory.

The syntax to use the chdir() method is given below.

Syntax

chdir("new-directory")

------------------------

import os

print("current working directory")

print(os.getcwd())

# Changing current directory with the new directiory

os.chdir("F:\\")

#It will display the current working directory

print(os.getcwd())

------------------

Deleting directory

The rmdir() method is used to delete the specified directory.

The syntax to use the rmdir() method is given below.

Syntax

os.rmdir(directory name)

-------------------------

import os

#removing the new directory

os.rmdir("C:\\Users\\BAIJU\\OneDrive\\Desktop\\kochi")

print("folder removed")

-------------------------------------------------

Python RegEx

A RegEx, or Regular Expression, is a sequence of characters that forms a

search pattern.

RegEx can be used to check if a string contains the specified search pattern.

RegEx Module

Python has a built-in package called re, which can be used to work with

Regular Expressions.

Import the re module:

import re

--------------------------------

RegEx in Python

When you have imported the re module, you can start using regular expressions:

eg:

Search the string to see if it starts with "The" and ends with "Spain":

-----------------------

RegEx Functions

The re module offers a set of functions that allows us to search a string

for a match:

findall:- Returns a list containing all matches

eg:

Print a list of all matches:

------------------------

import re

txt = "The rain in Spain"

x = re.findall("ai", txt)

print(x)

output:

['ai', 'ai']

--------------------

search:- Returns a Match object if there is a match

anywhere in the string

----------------

import re

txt = "The rain in Spain"

x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start())

output:

The first white-space character is located in position: 3

------------------

split: Returns a list where the string has been split at each match

Split at each white-space character:

----------------------

import re

txt = "The rain in Spain"

x = re.split("\s", txt)

print(x)

output:

['The', 'rain', 'in', 'Spain']

-------------------

sub: Replaces one or many matches with a string

Replace every white-space character with the number 9:

---------------------

import re

txt = "The rain in Spain"

x = re.sub("\s", "9", txt)

print(x)

output:

The9rain9in9Spain

-------------------

Metacharacters:

Metacharacters are characters with a special meaning:

[] A set of characters "[a-m]"

----------------

import re

txt = "The rain in Spain"

#Find all lower case characters alphabetically between "a" and "m":

x = re.findall("[a-m]", txt)

print(x)

output:

['h', 'e', 'a', 'i', 'i', 'a', 'i']

----------------------------

. Any character (except newline character) "he..o"

--------------------

#Search for a sequence that starts with "he", followed by

two (any) characters, and an "o":

----------------

import re

txt = "hello planet"

x = re.findall("he..o", txt)

print(x)

output:

['hello']

-------------------

^ Starts with "^hello"

----------

import re

txt = "hello planet"

#Check if the string starts with 'hello':

x = re.findall("^hello", txt)

if x:

print("Yes, the string starts with 'hello'")

else:

print("No match")

output:

Yes, the string starts with 'hello'

--------------------------------

$ Ends with "planet$"

-------------

import re

txt = "hello planet"

#Check if the string ends with 'planet':

x = re.findall("planet$", txt)

if x:

print("Yes, the string ends with 'planet'")

else:

print("No match")

output:

Yes, the string ends with 'planet'

-------------------------------------

\* Zero or more occurrences "he.\*o"

#Search for a sequence that starts with "he", followed by 0 or more

(any) characters, and an "o":

-----------------

import re

txt = "hello planet"

x = re.findall("he.\*o", txt)

print(x)

output:

['hello']

-------------------------

+ One or more occurrences "he.+o"

-----------

import re

txt = "hello planet"

#Search for a sequence that starts with "he", followed by 1

or more (any) characters, and an "o":

x = re.findall("he.+o", txt)

print(x)

output:

['hello']

--------------------------

? Zero or one occurrences "he.?o"

----------

import re

txt = "hello planet"

#Search for a sequence that starts with "he", followed by 0 or 1 (any)

character, and an "o":

x = re.findall("he.?o", txt)

print(x)

output:

[]

------------------

#This time we got no match, because there were not zero, not one, but

two characters

between "he" and the "o"

----------------------------

{} Exactly the specified number of occurrences "he.{2}o"

----------

import re

txt = "hello planet"

#Search for a sequence that starts with "he", followed excactly 2 (any)

characters, and an "o":

x = re.findall("he.{2}o", txt)

print(x)

output:

['hello']

-------------------------------

| Either or "falls|stays"

-------

import re

txt = "The rain in Spain falls mainly in the plain!"

#Check if the string contains either "falls" or "stays":

x = re.findall("falls|stays", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

['falls']

Yes, there is at least one match!

------------------------------

Sets

A set is a set of characters inside a pair of square brackets []

with a special meaning:

[arn] Returns a match where one of the specified characters (a, r, or n) is present

----------

import re

txt = "The rain in Spain"

#Check if the string has any a, r, or n characters:

x = re.findall("[arn]", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

['r', 'a', 'n', 'n', 'a', 'n']

Yes, there is at least one match!

------------------------------------

[a-n] Returns a match for any lower case character, alphabetically

between a and n

-----------

import re

txt = "The rain in Spain"

#Check if the string has any characters between a and n:

x = re.findall("[a-n]", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

['h', 'e', 'a', 'i', 'n', 'i', 'n', 'a', 'i', 'n']

Yes, there is at least one match!

----------------------------

[^arn] Returns a match for any character EXCEPT a, r, and n

------------

import re

txt = "The rain in Spain"

#Check if the string has other characters than a, r, or n:

x = re.findall("[^arn]", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

['T', 'h', 'e', ' ', 'i', ' ', 'i', ' ', 'S', 'p', 'i']

Yes, there is at least one match!

-----------------------------

[0123] Returns a match where any of the specified digits (0, 1, 2, or 3)

are present

------------

import re

txt = "The rain in Spain"

#Check if the string has any 0, 1, 2, or 3 digits:

x = re.findall("[0123]", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

[]

No match

-----------------------

[0-9] Returns a match for any digit between 0 and 9

------

import re

txt = "8 times before 11:45 AM"

#Check if the string has any digits:

x = re.findall("[0-9]", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

['8', '1', '1', '4', '5']

Yes, there is at least one match!

----------------------------

[0-5][0-9] Returns a match for any two-digit numbers from 00 and 59

-----------------

import re

txt = "8 times before 11:45 AM"

#Check if the string has any two-digit numbers, from 00 to 59:

x = re.findall("[0-5][0-9]", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

['11', '45']

Yes, there is at least one match!

-------------------------

[a-zA-Z] Returns a match for any character alphabetically between

a and z, lower case OR upper case

-------------

import re

txt = "8 times before 11:45 AM"

#Check if the string has any characters from a to z lower case, and A to Z upper case:

x = re.findall("[a-zA-Z]", txt)

print(x)

if x:

print("Yes, there is at least one match!")

else:

print("No match")

output:

['t', 'i', 'm', 'e', 's', 'b', 'e', 'f', 'o', 'r', 'e', 'A', 'M']

Yes, there is at least one match!

-------------------------------------------------------

Python OOPs Concepts

Like other general-purpose programming languages,

Python is also an object-oriented language since

its beginning. It allows us to develop applications

using an Object-Oriented approach. In Python, we

can easily create and use Like other general-purpose

programming languages, Python is also an

object-oriented language since its beginning. It

allows us to develop applications using an

Object-Oriented approach. In Python, we can easily

create and use classes and objects.classes and

objects. An object-oriented paradigm is to design

the program using classes and objects. The object

is related to real-word entities such as book,

house, pencil, etc. The oops concept focuses on

writing the reusable code. It is a widespread

technique to solve the problem by creating objects.

----------------

Major principles of object-oriented programming

system are given below.

Class

Object

Method

Inheritance

Polymorphism

Data Abstraction

Encapsulation

---------------------

Class

The class can be defined as a collection of objects.

It is a logical entity that has some specific

attributes and methods. For example: if you have

an employee class, then it should contain an

attribute and method, i.e. an email id, name, age,

salary, etc.

Object

The object is an entity that has state and

behavior. It may be any real-world object like

the mouse, keyboard, chair, table, pen, etc.

Method:

The method is a function that is associated with

an object. In Python, a method is not unique to

class instances. Any object type can have methods.

Inheritance:

Inheritance is the most important aspect of

object-oriented programming, which simulates

the real-world concept of inheritance. It

specifies that the child object acquires all

the properties and behaviors of the parent

object.

By using inheritance, we can create a class which

uses all the properties and behavior of another

class. The new class is known as a derived class

or child class, and the one whose properties are

acquired is known as a base class or parent class.

It provides the re-usability of the code.

Polymorphism

Polymorphism contains two words "poly" and "morphs".

Poly means many, and morph means shape. By

polymorphism, we understand that one task can be

performed in different ways. For example - you have

a class animal, and all animals speak. But they

speak differently. Here, the "speak" behavior is

polymorphic in a sense and depends on the animal.

So, the abstract "animal" concept does not actually

"speak", but specific animals (like dogs and cats)

have a concrete implementation of the action

"speak".

Encapsulation:

Encapsulation is also an essential aspect of

object-oriented programming. It is used to

restrict access to methods and variables. In

encapsulation, code and data are wrapped together

within a single unit from being modified by

accident.

Data Abstraction

Data abstraction and encapsulation both are often

used as synonyms. Both are nearly synonyms because

data abstraction is achieved through encapsulation.

Abstraction is used to hide internal details and

show only functionalities. Abstracting something

means to give names to things so that the name

captures the core of what a function or a whole

program does.

--------------------

Object-oriented vs. Procedure-oriented Programming

languages:

->Object-oriented programming is the problem-solving

approach and used where computation is done by

using objects.

->Procedural programming uses a list of instructions

to do computation step by step.

-------------------

->OOPs makes the development and maintenance easier.

->In procedural programming, It is not easy to

maintain the codes when the project becomes lengthy.

-----------------

->OOPs simulates the real world entity. So

real-world problems can be easily solved through

oops.

->Procedural doesn't simulate the real world. It

works on step by step instructions divided into

small parts called functions.

-----------------------

->It provides data hiding. So it is more secure

than procedural languages. You cannot access

private data from anywhere.

->Procedural language doesn't provide any proper

way for data binding, so it is less secure.

-------------------

->Example of object-oriented programming languages

is C++, Java, .Net, Python, C#, etc.

Example of procedural languages are: C, Fortran,

Pascal, VB etc.

--------------------------

Python Class and Objects

Suppose a class is a prototype of a building. A building contains all

the details about the floor, rooms, doors, windows, etc. we can make

as many buildings as we want, based on these details. Hence, the building

can be seen as a class, and we can create as many objects of this class.

On the other hand, the object is the instance of a class. The process of

creating an object can be called instantiation.

------------------------------

Creating classes in Python

In Python, a class can be created by using the keyword class,

followed by the class name. The syntax to create a class is given below.

Syntax:

class ClassName:

#statement\_suite

A class contains a statement suite including

fields, constructor, function, etc. definition.

-------------

To create a class Employee which contains two

fields as Employee id, and name. The class also

contains a function display(), which is used to

display the information of the Employee.

class Employee:

id = 10

name = "Devansh"

def display(self):

print(self.id,self.name)

------------------------------

Here, the self is used as a reference variable,

which refers to the current class object. It is

always the first argument in the function

definition. However, using self is optional in

the function call.

The self-parameter

The self-parameter refers to the current instance

of the class and accesses the class variables.

We can use anything instead of self,but it must

be the first parameter of any function which

belongs to the class.

Creating an instance of the class

A class needs to be instantiated if we want to

use the class attributes in another class or

method. A class can be instantiated by calling

the class using the class name.

The syntax to create the instance of the class

is given below.

<object-name> = <class-name>(<arguments>)

---------------------------

#Create the instance of the class Employee

class Employee:

id = 10

name = "John"

def display(self):

print("ID: %d \nName: %s"%(self.id,self.name))

# Creating a emp instance of Employee class

emp = Employee()

emp.display()

output:

ID: 10

Name: John

-------------------------------

we have created the Employee class which has two

attributes named id and name and assigned value

to them. We can observe we have passed the self

as parameter in display function. It is used to

refer to the same class attribute.

We have created a new instance object named emp.

By using it,

we can access the attributes of the class.

--------------------

Delete the Object

We can delete the properties of the object or

object itself by using the del keyword.

---------------------

class Employee:

id = 10

name = "John"

def display(self):

print("ID: %d \nName: %s" % (self.id, self.name))

# Creating a emp instance of Employee class

emp = Employee()

emp.display()

# Deleting the property of object

del emp.id

# Deleting the object itself

del emp

emp.display() #error

--------------------

Python Constructor:

A constructor is a special type of method

(function) which is used to initialize the

instance members of the class.

Constructors can be of two types.

1. Parameterized Constructor

2. Non-parameterized Constructor

Constructor definition is executed when we create

the object of this class.

Creating the constructor in python

In Python, the method the \_\_init\_\_() simulates

the constructor of the class.

This method is called when the class is

instantiated. It accepts the self-keyword as a

first argument which allows accessing the

attributes or method of the class.

We can pass any number of arguments at the time

of creating the class object, depending upon

the \_\_init\_\_() definition. It is mostly used to

initialize the class attributes. Every class

must have a constructor, even if it simply

relies on the default constructor.

---------------------------

class Employee:

def \_\_init\_\_(self, name1, id1):

self.id = id1

self.name = name1

def display(self):

print("ID: %d \nName: %s" % (self.id, self.name))

emp1 = Employee("John", 101)

emp2 = Employee("David", 102)

# accessing display() method to print employee 1 information

emp1.display()

# accessing display() method to print employee 2 information

emp2.display()

output:

ID: 101

Name: John

ID: 102

Name: David

----------

#Counting the number of objects of a class:

class Student:

count = 0

def \_\_init\_\_(self):

Student.count = Student.count + 1

s1=Student()

s2=Student()

s3=Student()

print("The number of students:",Student.count)

output:

The number of students: 3

-------------------------------

Python Non-Parameterized Constructor

The non-parameterized constructor uses when we do not want to manipulate

the value or the constructor that has only self as an argument.

----------------------

class Student:

# Constructor - non parameterized

def \_\_init\_\_(self):

print("This is non parametrized constructor")

def show(self,name):

print("Hello",name)

student = Student()

student.show("John")

output:

This is non parametrized constructor

Hello John

--------------------

Python Parameterized Constructor

The parameterized constructor has multiple parameters along with the self.

------------------------

class Student:

# Constructor - parameterized

def \_\_init\_\_(self, name):

print("This is parametrized constructor")

self.name = name

def show(self):

print("Hello",self.name)

student = Student("John")

student.show()

output:

This is parametrized constructor

Hello John

------------------------

Python Default Constructor

When we do not include the constructor in the class or forget to declare it,

then that becomes the default constructor. It does not perform any task but

initializes the objects.

-----------------------

class Student:

roll\_num = 101

name = "Joseph"

def display(self):

print(self.roll\_num,self.name)

st = Student()

st.display()

output:

101 Joseph

---------------------

Python built-in class functions

The built-in functions defined in the class are described in the following table.

1. getattr(obj,name) It is used to access the attribute of the object.

2. setattr(obj, name,value) It is used to set a particular value to the specific

attribute of an object.

3. delattr(obj, name) It is used to delete a specific attribute.

4. hasattr(obj, name) It returns true if the object contains some specific

attribute.

-------------------------------

class Student:

def \_\_init\_\_(self, name, id, age):

self.name = name

self.id = id

self.age = age

# creates the object of the class Student

s = Student("John", 101, 22)

# prints the attribute name of the object s

print(getattr(s, 'name'))

# reset the value of attribute age to 23

setattr(s, "age", 23)

# prints the modified value of age

print(getattr(s, 'age'))

# prints true if the student contains the attribute with name id

print(hasattr(s, 'id'))

# deletes the attribute age

delattr(s, 'age')

# this will give an error since the attribute age has been deleted

print(s.age)

-------------------------

Built-in class attributes

Along with the other attributes, a Python class

also contains some built-in class attributes

which provide information about the class.

----------------

1. \_\_dict\_\_ It provides the dictionary containing

the information about the class namespace.

2. \_\_doc\_\_ It contains a string which has the class documentation

3. \_\_name\_\_ It is used to access the class name.

4. \_\_module\_\_ It is used to access the module in which, this class is defined.

5. \_\_bases\_\_ It contains a tuple including all base classes.

-------------------------------

class Student:

"A class about students and their addresses"

def \_\_init\_\_(self,name,id,age):

self.name = name

self.id = id

self.age = age

def display\_details(self):

print("Name:%s, ID:%d, age:%d" % (self.name,self.id,self.age))

s = Student("John",101,22)

print("Doc:",s.\_\_doc\_\_)

print("Dict:",s.\_\_dict\_\_)

print("Module:",s.\_\_module\_\_)

print("Name:",Student.\_\_name\_\_)

print("Base:",Student.\_\_bases\_\_)

Output:

Doc: A class about students and their addresses

Dict: {'name': 'John', 'id': 101, 'age': 22}

Module: \_\_main\_\_

Name: Student

Base: (<class 'object'>,)

-----------------------------

Python Inheritance:

Inheritance is an important aspect of the

object-oriented paradigm. Inheritance provides

code reusability to the program because we can

use an existing class to create a new class instead

of creating it from scratch.

In inheritance, the child class acquires the

properties and can access all the data members

and functions defined in the parent class. A child

class can also provide its specific implementation

to the functions of the parent class.

In python, a derived class can inherit base class

by just mentioning the base in the bracket after

the derived class name.

--------------

Single Inheritance

Syntax

class derived-class(base class):

<class-suite>

----------------------------

# Single level inheritance

class Animal:

def speak(self):

print("Animal Speaking")

#child class Dog inherits the base class Animal

class Dog(Animal):

def bark(self):

print("dog barking")

d = Dog()

d.bark()

d.speak()

---------------------

Python Multi-Level inheritance

Multi-Level inheritance is possible in python like

other object-oriented languages. Multi-level

inheritance is archived when a derived class

inherits another derived class. There is no limit

on the number of levels up to which, the multi-level

inheritance is archived in python.

The syntax of multi-level inheritance is given below.

Syntax

class class1:

<class-suite>

class class2(class1):

<class suite>

class class3(class2):

<class suite>

.

.

-----------------------

#multi level inheritance

class Animal:

def speak(self):

print("Animal Speaking")

#The child class Dog inherits the base class Animal

class Dog(Animal):

def bark(self):

print("dog barking")

#The child class Dogchild inherits another child class Dog

class DogChild(Dog):

def eat(self):

print("Eating bread...")

d = DogChild()

d.bark()

d.speak()

d.eat()

---------------------------

Python Multiple inheritance:

Python provides us the flexibility to inherit multiple base classes

in the child class.

The syntax to perform multiple inheritance is given below.

Syntax

class Base1:

<class-suite>

class Base2:

<class-suite>

.

.

.

class BaseN:

<class-suite>

class Derived(Base1, Base2, ...... BaseN):

<class-suite>

-----------------------

class Calculation1:

def Summation(self,a,b):

return a+b

class Calculation2:

def Multiplication(self,a,b):

return a\*b

class Derived(Calculation1,Calculation2):

def Divide(self,a,b):

return a/b

d = Derived()

print(d.Summation(10,20))

print(d.Multiplication(10,20))

print(d.Divide(10,20))

-------------------------

The issubclass(sub,sup) method

The issubclass(sub, sup) method is used to check the relationships

between the specified classes. It returns true if the first class

is the subclass of the second class, and false otherwise.

--------------------------------------

class Calculation1:

def Summation(self,a,b):

return a+b;

class Calculation2:

def Multiplication(self,a,b):

return a\*b;

class Derived(Calculation1,Calculation2):

def Divide(self,a,b):

return a/b;

d = Derived()

print(issubclass(Derived,Calculation2))

print(issubclass(Calculation1,Calculation2))

--------------------------------

The isinstance(obj, class) method

The isinstance() method is used to check the relationship between the

objects and classes. It returns true if the first parameter, i.e.,

obj is the instance of the second parameter, i.e., class.

------------------------------------------

class Calculation1:

def Summation(self,a,b):

return a+b;

class Calculation2:

def Multiplication(self,a,b):

return a\*b;

class Derived(Calculation1,Calculation2):

def Divide(self,a,b):

return a/b;

d = Derived()

print(isinstance(d,Derived))

----------------------------

Method Overriding

We can provide some specific implementation of the parent class method

in our child class. When the parent class method is defined in the

child class with some specific implementation, then the concept is

called method overriding. We may need to perform method overriding

in the scenario where the different definition of a parent class

method is needed in the child class.

------------------------------------

class Animal:

def speak(self):

print("speaking")

class Dog(Animal):

def speak(self):

print("Barking")

d = Dog()

d.speak()

-------------------------

#Real Life Example of method overriding

class Bank:

def getroi(self):

return 10

class SBI(Bank):

def getroi(self):

return 7

class ICICI(Bank):

def getroi(self):

return 8

b1 = Bank()

b2 = SBI()

b3 = ICICI()

print("Bank Rate of interest:",b1.getroi())

print("SBI Rate of interest:",b2.getroi())

print("ICICI Rate of interest:",b3.getroi())

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------------------

Database

What is Data?

Data is a collection of a distinct small unit of

information. It can be used in a variety of forms

like text, numbers, media, bytes, etc. it can be

stored in pieces of paper or electronic memory, etc.

What is Database?

A database is an organized collection of data, so

that it can be easily accessed and managed.

You can organize data into tables, rows, columns,

and index it to make it easier to find relevant

information.

Database handlers create a database in such a way

that only one set of software program provides

access of data to all the users.

The main purpose of the database is to operate a

large amount of information by storing, retrieving,

and managing data.

There are many dynamic websites on the World Wide

Web nowadays which are handled through databases.

For example, a model that checks the availability

of rooms in a hotel. It is an example of a dynamic

website that uses a database.

There are many databases available like MySQL,

Sybase, Oracle, Informix, PostgreSQL, SQL Server,etc.

Modern databases are managed by the database

management system (DBMS).

SQL or Structured Query Language is used to operate

on the data stored in a database. SQL depends on

relational algebra and tuple relational calculus.

Evolution of Databases

The database has completed more than 50 years of

journey of its evolution from flat-file system to

relational and objects relational systems.

It has gone through several generations.

The Evolution

File-Based

1968 was the year when File-Based database were

introduced. In file-based databases, data was

maintained in a flat file. Though files have many

advantages, there are several limitations.

One of the major advantages is that the file system

has various access methods,

e.g., sequential, indexed, and random.

RDBMS (Relational Database Management System)

The word RDBMS is termed as 'Relational Database

Management System.' It is represented as a table

that contains rows and column.

What is table/Relation?

Everything in a relational database is stored in

the form of relations. The RDBMS database uses

tables to store data.

A table is a collection of related data entries

and contains rows and columns to store data.

Each table represents some real-world objects such

as person, place, or event about which information

is collected. The organized collection of data

into a relational table is known as the logical

view of the database. Properties of a Relation:

Each relation has a unique name by which it is

identified in the database. Relation does not

contain duplicate tuples. The tuples of a relation

have no specific order.

Student table.

ID Name AGE COURSE

1 Ajeet 24 B.Tech

2 aryan 20 C.A

3 Mahesh 21 BCA

4 Ratan 22 MCA

5 Vimal 26 BSC

What is a row or record?

A row of a table is also called a record or tuple.

It contains the specific information of each entry

in the table. It is a horizontal entity in the

table.

Let's see one record/row in the table.

ID Name AGE COURSE

1 Ajeet 24 B.Tech

What is a column/attribute?

A column is a vertical entity in the table which

contains all information associated with a

specific field in a table. For example, "name"

is a column in the above table which contains

all information about a student's name.

Name

Ajeet

Aryan

Mahesh

Ratan

Vimal

What is data item/Cells?

The smallest unit of data in the table is the

individual data item. It is stored at the

intersection of tuples and attributes.

SQL:

SQL is a database computer language designed for

the retrieval and management of data in a

relational database.

SQL- stands for Structured Query Language.

Why to Learn SQL?

SQL is Structured Query Language, which is a

computer language for storing, manipulating and

retrieving data stored in a relational database.

SQL is the standard language for Relational

Database System. All the Relational Database

Management Systems (RDMS) like MySQL, MS Access,

Oracle, Sybase, Informix, Postgres and SQL Server

use SQL as their standard database language.

Applications of SQL

As mentioned before, SQL is one of the most widely

used query language over the databases. I'm going

to list few of them here:

Allows users to access data in the relational

database management systems.

Allows users to describe the data.

Allows users to define the data in a database

and manipulate that data.

Allows to embed within other languages using SQL

modules, libraries & pre-compilers.

Allows users to create and drop databases and

tables.

SQL - Overview

SQL is a language to operate databases; it includes

database creation, deletion, fetching rows,

modifying rows, etc.

What is SQL?

SQL is Structured Query Language, which is a

computer language for storing, manipulating and

retrieving data stored in a relational database.

SQL is the standard language for Relational

Database System. All the Relational Database

Management Systems (RDMS) like MySQL, MS Access,

Oracle, Sybase, Informix, Postgres and SQL Server

use SQL as their standard database language.

-----------------------

SQL Commands

The standard SQL commands to interact with

relational databases are CREATE, SELECT, INSERT,

UPDATE, DELETE and DROP. These commands can be

classified into the following groups based on

their nature −

DDL - Data Definition Language

Sr.No. Command & Description

1

CREATE

Creates a new table, a view of a table, or other object

in the database.

2

ALTER

Modifies an existing database object, such as a table.

3

DROP

Deletes an entire table, a view of a table or other objects

in the database.

--------------------

DML - Data Manipulation Language

Sr.No. Command & Description

1

SELECT

Retrieves certain records from one or more tables.

2

INSERT

Creates a record.

3

UPDATE

Modifies records.

4

DELETE

Deletes records.

-----------------

DCL - Data Control Language

Sr.No. Command & Description

1

GRANT

Gives a privilege to user.

2

REVOKE

Takes back privileges granted from user.

------------

SQL - RDBMS Concepts

What is RDBMS?

RDBMS stands for Relational Database Management

System. RDBMS is the basis for SQL, and for all

modern database systems like MS SQL Server,

IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS)

is a database management system (DBMS) that is

based on the relational model as introduced

by E. F. Codd.

What is a table?

The data in an RDBMS is stored in database objects

which are called as tables. This table is

basically a collection of related data entries

and it consists of numerous columns and rows.

a CUSTOMERS table −

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

What is a field?

Every table is broken up into smaller entities

called fields. The fields in the CUSTOMERS table

consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to

maintain specific information about every record

in the table.

What is a Record or a Row?

A record is also called as a row of data is each

individual entry that exists in a table. For

example, there are 7

records in the above

CUSTOMERS table. Following is a single row of data

or record in the CUSTOMERS table −

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

+----+----------+-----+-----------+----------+

What is a column?

A column is a vertical entity in a table that

contains all information associated with a

specific field in a table.

For example, a column in the CUSTOMERS table is

ADDRESS, which represents location description

and would be as shown below −

+-----------+

| ADDRESS |

+-----------+

| Ahmedabad |

| Delhi |

| Kota |

| Mumbai |

| Bhopal |

| MP |

| Indore |

+----+------+

What is a NULL value?

A NULL value in a table is a value in a field that

appears to be blank, which means a field with a

NULL value is a field with no value.

It is very important to understand that a NULL

value is different than a zero value or a field

that contains spaces. A field with a NULL value

is the one that has been left blank during a

record creation.

SQL Constraints

Constraints are the rules enforced on data columns

on a table. These are used to limit the type of

data that can go into a table. This ensures the

accuracy and reliability of the data in the database.

Constraints can either be column level or table level.

Column level constraints are applied only to one column

whereas, table level constraints

are applied to the entire table.

------------

SQL - RDBMS Databases

MySQL

MySQL is an open source SQL database, which is developed

by a Swedish company – MySQL AB. MySQL is pronounced as

"my ess-que-ell," in contrast with SQL, pronounced "sequel."

MySQL is supporting many different platforms including

Microsoft Windows, the major Linux distributions, UNIX,

and Mac OS X.

Features

High Performance.

High Availability.

Scalability and Flexibility Run anything.

Robust Transactional Support.

Web and Data Warehouse Strengths.

Strong Data Protection.

Comprehensive Application Development.

Management Ease.

Open Source Freedom and 24 x 7 Support.

Lowest Total Cost of Ownership.

-------------------------------------

MySQL uses many different data types broken into

three categories −

Numeric

Date and Time

String Types.

Numeric Data Types

MySQL uses all the standard ANSI SQL numeric data

types.

The following list shows the common numeric data

types and their descriptions −

INT − A normal-sized integer that can be signed

or unsigned. If signed, the allowable range is

from -2147483648 to 2147483647. If unsigned, the

allowable range is from 0 to 4294967295. You can

specify a width of up to 11 digits.

TINYINT − A very small integer that can be signed

or unsigned. If signed, the allowable range is

from -128 to 127. If unsigned, the allowable range

is from 0 to 255. You can specify a width of up

to 4 digits.

SMALLINT − A small integer that can be signed or

unsigned. If signed, the allowable range is from

-32768 to 32767. If unsigned, the allowable range

is from 0 to 65535. You can specify a width of

up to 5 digits.

MEDIUMINT − A medium-sized integer that can be

signed or unsigned. If signed, the allowable range

is from -8388608 to 8388607. If unsigned, the

allowable range is from 0 to 16777215. You can

specify a width of up to 9 digits.

BIGINT − A large integer that can be signed or

unsigned. If signed, the allowable range is

from -9223372036854775808 to 9223372036854775807.

If unsigned, the allowable range is from

0 to 18446744073709551615. You can specify a

width of up to 20 digits.

FLOAT(M,D) − A floating-point number that cannot

be unsigned. You can define the display length

(M) and the number of decimals (D). This is not

required and will default to 10,2, where 2 is

the number of decimals and 10 is the total

number of digits (including decimals). Decimal

precision can go to 24 places for a FLOAT.

DOUBLE(M,D) − A double precision floating-point

number that cannot be unsigned. You can define

the display length (M) and the number of decimals

(D). This is not required and will default to

16,4, where 4 is the number of decimals. Decimal

precision can go to 53 places for a DOUBLE. REAL

is a synonym for DOUBLE.

DECIMAL(M,D) − An unpacked floating-point number

that cannot be unsigned. In the unpacked decimals,

each decimal corresponds to one byte. Defining

the display length (M) and the number of decimals

(D) is required. NUMERIC is a synonym for DECIMAL.

-----------------

Date and Time Types

The MySQL date and time datatypes are as follows −

DATE − A date in YYYY-MM-DD format, between

1000-01-01 and 9999-12-31. For example, December

30th, 1973 would be stored as 1973-12-30.

DATETIME − A date and time combination in

YYYY-MM-DD HH:MM:SS format, between

1000-01-01 00:00:00 and 9999-12-31 23:59:59.

For example, 3:30 in the afternoon on December

30th, 1973 would be stored as 1973-12-30 15:30:00.

TIMESTAMP − A timestamp between midnight,

January 1st, 1970 and sometime in 2037. This

looks like the previous DATETIME format, only

without the hyphens between numbers; 3:30 in

the afternoon on December 30th, 1973

would be stored as 19731230153000

( YYYYMMDDHHMMSS ).

TIME − Stores the time in a HH:MM:SS format.

-------------

String Types

Although the numeric and date types are fun,

most data you'll store will be in a string format.

This list describes the common string datatypes

in MySQL.

CHAR(M) − A fixed-length string between 1 and 255

characters in length (for example CHAR(5)),

right-padded with spaces to the specified length

when stored. Defining a length is not required,

but the default is 1.

VARCHAR(M) − A variable-length string between 1

and 255 characters in length. For example,

VARCHAR(25). You must define a length when

creating a VARCHAR field.

BLOB or TEXT − A field with a maximum length of

65535 characters. BLOBs are "Binary Large Objects"

and are used to store large amounts of binary

data, such as images or other types of files.

Fields defined as TEXT also hold large amounts

of data. The difference between the two is that

the sorts and comparisons on the stored data are

case sensitive on BLOBs and are not case sensitive

in TEXT fields. You do not specify a

length with BLOB or TEXT.

TINYBLOB or TINYTEXT − A BLOB or TEXT column with

a maximum length of 255 characters. You do not

specify a length with TINYBLOB or TINYTEXT.

MEDIUMBLOB or MEDIUMTEXT − A BLOB or TEXT column

with a maximum length of 16777215 characters.

You do not specify a length with MEDIUMBLOB or

MEDIUMTEXT.

LONGBLOB or LONGTEXT − A BLOB or TEXT column

with a maximum length of 4294967295 characters.

You do not specify a length with LONGBLOB or

LONGTEXT.

------------------

SQL - Operators

What is an Operator in SQL?

An operator is a reserved word or a character

used primarily in an SQL statement's WHERE clause

to perform operation(s), such as comparisons and

arithmetic operations. These Operators are used

to specify conditions in an SQL statement and

to serve as conjunctions for multiple

conditions in a statement.

Arithmetic operators

Comparison operators

Logical operators

Operators used to negate conditions

-----------

SQL Arithmetic Operators

Assume 'variable a' holds 10 and 'variable b' holds

20, then −

+ (Addition) Adds values on either side of

the operator. a + b will give 30

- (Subtraction) Subtracts right hand operand from

left hand operand. a - b will give -10

\* (Multiplication) Multiplies values on either

side of the operator. a \* b will give 200

/ (Division) Divides left hand operand by

right hand operand. b / a will give 2

% (Modulus) Divides left hand operand by right

hand operand and returns remainder. b % a

will give 0

-----------------

SQL Comparison Operators

Assume 'variable a' holds 10 and 'variable b'

holds 20, then Operator Description Example

= Checks if the values of two operands are

equal or not, if yes then condition becomes

true. (a = b) is not true.

!= Checks if the values of two operands are

equal or not, if values are not equal then

condition becomes true. (a != b) is true.

<> Checks if the values of two operands are

equal or not, if values are not equal then

condition becomes true. (a <> b) is true.

> Checks if the value of left operand is

greater than the value of right operand, if

yes then condition becomes true.

(a > b) is not true.

< Checks if the value of left operand is

less than the value of right operand, if yes

then condition becomes true.

(a < b) is true.

>= Checks if the value of left operand is

greater than or equal to the value of right

operand, if yes then condition becomes true.

(a >= b) is not true.

<= Checks if the value of left operand is

less than or equal to the value of right operand,

if yes then condition becomes true.

(a <= b) is true.

!< Checks if the value of left operand is not

less than the value of right operand, if yes

then condition becomes

true. (a !< b) is false

SQL Logical Operators

Here is a list of all the logical operators available in

SQL.

Sr.No. Operator & Description

1

ALL

The ALL operator is used to compare a value to all values

in another value set.

2

AND

The AND operator allows the existence of multiple

conditions in an SQL statement's

WHERE clause.

3

ANY

The ANY operator is used to compare a value to any

applicable value in the list as per the condition.

4

BETWEEN

The BETWEEN operator is used to search for values

that are within a set of values, given the

minimum value and the

maximum value.

5

EXISTS

The EXISTS operator is used to search for the

presence of a row in a specified table that

meets a certain criterion.

6

IN

The IN operator is used to compare a value to

a list of literal values that have been specified.

7

LIKE

The LIKE operator is used to compare a value to

similar values using wildcard operators.

8

NOT

The NOT operator reverses the meaning of the

logical operator with which it is used.

Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This

is a negate operator.

9

OR

The OR operator is used to combine multiple

conditions in an SQL statement's WHERE clause.

10

IS NULL

The NULL operator is used to compare a value with

a NULL value.

11

UNIQUE

The UNIQUE operator searches every row of a

specified table for uniqueness (no duplicates).

--------------

SQL - CREATE Database

The SQL CREATE DATABASE statement is used to

create a new SQL database.

Syntax

The basic syntax of this CREATE DATABASE statement

is as follows −

CREATE DATABASE DatabaseName;

If you want to create a new database <testDB>,

then the

CREATE DATABASE statement would be as shown

below −

mysql> CREATE DATABASE testDB;

Make sure you have the admin privilege before creating any

database. Once a database is created, you can check it in

the list of databases as follows −

mysql> SHOW DATABASES;

-------------------

SQL - DROP or DELETE Database

The SQL DROP DATABASE statement is used to drop an existing

database in SQL schema.

Syntax

The basic syntax of DROP DATABASE statement is as

follows −

DROP DATABASE DatabaseName;

If you want to delete an existing database <testDB>,

then

the DROP DATABASE statement would be as shown

below −

mysql> DROP DATABASE testDB;

Then,

mysql> SHOW DATABASES;

--------------------------------------

SQL - SELECT Database, USE Statement

When you have multiple databases in your SQL Schema,

then before starting your operation, you would need

to select a database where all the operations would

be performed.

The SQL USE statement is used to select any existing

database in the SQL schema.

Syntax

The basic syntax of the USE statement is as shown below −

USE DatabaseName;

You can check the available databases as shown below −

First create a database named smec

mysql> CREATE DATABASE smec;

for selecting

mysql> use smec;

Database changed

mysql>

mysql> SHOW DATABASES;

Now, you can execute the following SQL command .

mysql>create database kochi;

mysql> SHOW DATABASES;

Select A Database:

use database\_name

mysql> USE kochi;

List tables in the database

mysql>show tables;

----------------------------

SQL - CREATE Table

Creating a basic table involves naming the table and

defining its columns and each column's data type.

The SQL CREATE TABLE statement is used to create a

new table.

Syntax

The basic syntax of the CREATE TABLE statement is

as follows −

CREATE TABLE table\_name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY( one or more columns )

);

-------------

CREATE TABLE is the keyword telling the database

system what you want to do. In this case, you want

to create a new table. The unique name or identifier

for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column

in the table and what sort of data type it is.

creates a CUSTOMERS table with an ID as a primary

key and NOT NULL are the constraints showing that

these fields cannot be NULL while creating

records in this table −

mysql> CREATE TABLE CUSTOMERS(ID INT NOT NULL,NAME VARCHAR(20) NOT NULL,AGE INT NOT NULL,ADDRESS CHAR(25) ,SALARY DECIMAL(18, 2),PRIMARY KEY (ID));

You can verify if your table has been created

successfully by looking at the message displayed

by the SQL server, otherwise you can use the

DESC command as follows −

syntax:

desc table\_name

mysql> DESC CUSTOMERS;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int | NO | PRI | NULL | |

| NAME | varchar(20) | NO | | NULL | |

| AGE | int | NO | | NULL | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

--------------------------------

SQL Constraints

Constraints are the rules enforced on data columns

on a table. These are used to limit the type of data

that can go into a table. This ensures the accuracy

and reliability of the data in the database.

Constraints can either be column level or table

level. Column level constraints are applied only

to one column whereas, table level constraints are

applied to the entire table.

SQL - NOT NULL Constraint

By default, a column can hold NULL values. If you do

not want a column to have a NULL value, then you need

to define such a constraint on this column specifying

that NULL is now not allowed for that column.

A NULL is not the same as no data, rather, it represents

unknown data.

creates a new table called CUSTOMERS and adds five

columns, three of which,

are ID NAME and AGE, In this we specify not to accept

NULLs −

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

-------------

Droping a table

Syntax:

drop table tablename

mysql>show tables;

mysql>drop table CUSTOMERS;

mysql>show tables;

------------------------

SQL - DEFAULT Constraint

The DEFAULT constraint provides a default value to a

column when the INSERT

INTO statement does not provide a specific value.

creates a new table called CUSTOMERS and adds five

columns. Here, the SALARY column is set to 5000.00

by default, so in case the INSERT INTO statement

does not provide a value for this column, then by

default this column would be set to 5000.00.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(25) ,

SALARY DECIMAL(18, 2) DEFAULT 5000.00,

PRIMARY KEY (ID)

);

-----------------------------------

SQL - UNIQUE Constraint

The UNIQUE Constraint prevents two records from having

identical values in a column. In the CUSTOMERS table,

for example, you might want to prevent two

or more people from having an identical age.

creates a new table called CUSTOMERS and adds five

columns. Here, the AGE column is set to UNIQUE, so

that you cannot have two records with the same age.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR(20) NOT NULL,

AGE INT NOT NULL UNIQUE,

ADDRESS CHAR(25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

--------------------------

SQL - Primary Key

A primary key is a field in a table which uniquely

identifies each row/record in a database table.

Primary keys must contain unique values.

A primary key column cannot have NULL values.

A table can have only one primary key, which may

consist of single or multiple fields. When multiple

fields are used as a primary key, they are called a

composite key.

Create Primary Key

Here is the syntax to define the ID attribute as a

primary key in a

CUSTOMERS table.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

-----------------

SQL - Foreign Key

A foreign key is a key used to link two tables

together. This is sometimes also called as a

referencing key.

A Foreign Key is a column or a combination of

columns whose values match a

Primary Key in a different table.

The relationship between 2 tables matches the Primary

Key in one of the tables

with a Foreign Key in the second table.

Consider the structure of the following two tables.

create CUSTOMERS table

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

create ORDERS table:

CREATE TABLE ORDERS (

ID INT NOT NULL,

DATE DATETIME,

CUSTOMER\_ID INT references CUSTOMERS(ID),

AMOUNT double,

PRIMARY KEY (ID)

);

-----------------------------

SQL - CHECK Constraint

The CHECK Constraint enables a condition to check the value being entered into

a record. If the condition evaluates to false, the record violates the

constraint and isn't entered the table.

creates a new table called CUSTOMERS and adds five columns. Here, we add a

CHECK with AGE column, so that you cannot have any CUSTOMER who is

below 18 years.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL CHECK (AGE >= 18),

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

--------------------

MySQL ALTER Table

MySQL ALTER statement is used when you want to

change the name of your table or any table field.

It is also used to add or delete an existing column

in a table.

The ALTER statement is always used with "ADD",

"DROP" and "MODIFY" commands according to the

situation.

1) ADD a column in the table

Syntax:

ALTER TABLE table\_name ADD new\_column\_name column\_definition

[ FIRST | AFTER column\_name ];

Parameters

table\_name: It specifies the name of the table that

you want to modify.

new\_column\_name: It specifies the name of the new

column that you want to add to the table.

column\_definition: It specifies the data type and

definition of the column (NULL or NOT NULL, etc).

FIRST | AFTER column\_name: It is optional. It tells

MySQL where in the table to create the column. If

this parameter is not specified, the new column will

be added to the end of the table.

mysql> desc customers;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int | NO | PRI | NULL | |

| name | varchar(20) | YES | | NULL | |

| AGE | int | NO | | NULL | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

mysql>alter table customers add mob varchar(12);

mysql> desc customers;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int | NO | PRI | NULL | |

| name | varchar(20) | YES | | NULL | |

| AGE | int | NO | | NULL | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

| mob | varchar(12) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

----------------

Add multiple columns in the table

ALTER TABLE table\_name

ADD new\_column\_name column\_definition

[ FIRST | AFTER column\_name ],

ADD new\_column\_name column\_definition

[ FIRST | AFTER column\_name ],

-----------------------

MODIFY column in the table

The MODIFY command is used to change the column

definition of the table.

Syntax:

ALTER TABLE table\_name MODIFY column\_name

column\_definition [ FIRST | AFTER column\_name ];

mysql> alter table customers modify name varchar(25);

Query OK, 0 rows affected (0.14 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc customers;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int | NO | PRI | NULL | |

| name | varchar(25) | YES | | NULL | |

| AGE | int | NO | | NULL | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL |

| mob | varchar(12) | YES | | NULL |

+---------+---------------+------+-----+---------+-------+

------------------

DROP column in table

Syntax:

ALTER TABLE table\_name DROP COLUMN column\_name;

mysql> alter table customers drop column mob;

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc customers;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int | NO | PRI | NULL | |

| name | varchar(25) | YES | | NULL | |

| AGE | int | NO | | NULL | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

RENAME column in table

Syntax:

ALTER TABLE table\_name CHANGE COLUMN old\_name new\_name

column\_definition

[ FIRST | AFTER column\_name ]

mysql> ALTER TABLE customers CHANGE COLUMN name names varchar(25);

Query OK, 0 rows affected (0.01 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> desc customers;

+---------+---------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+---------+---------------+------+-----+---------+-------+

| ID | int | NO | PRI | NULL | |

| names | varchar(25) | YES | | NULL | |

| AGE | int | NO | | NULL | |

| ADDRESS | char(25) | YES | | NULL | |

| SALARY | decimal(18,2) | YES | | NULL | |

+---------+---------------+------+-----+---------+-------+

----------------------

RENAME table

Syntax:

ALTER TABLE table\_name RENAME TO new\_table\_name;

mysql> ALTER TABLE orders RENAME TO ekm;

mysql>show tables;

--------------------------------

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

-------------------------------

SQL - INSERT Query

The SQL INSERT INTO Statement is used to add new

rows of data to a table in the database.

Syntax-1

There are two basic syntaxes of the INSERT INTO

statement which are shown below.

INSERT INTO TABLE\_NAME (column1, column2, column3,...columnN)

VALUES (value1, value2, value3,...valueN);

Here, column1, column2, column3,...columnN are the

names of the columns in the table into which you

want to insert the data.

You may not need to specify the column(s) name in

the SQL query if you are adding values for all the

columns of the table. But make sure the order of

the values is in the same order as the columns

The SQL INSERT INTO syntax will be as follows −

Syntax-2:

INSERT INTO TABLE\_NAME VALUES (value1,value2,value3,...valueN);

The following statements would create six records in the CUSTOMERS table. in the table.

mysql>INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

mysql>INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );

mysql>INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );

mysql>INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );

mysql>INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );

mysql>INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (6, 'Komal', 22, 'MP', 4500.00 );

You can create a record in the CUSTOMERS table by using the second syntax as shown below.

mysql>INSERT INTO CUSTOMERS VALUES (7, 'Muffy', 24, 'Indore', 10000.00 );

------------

SQL - SELECT Query

The SQL SELECT statement is used to fetch the data from a

database table which returns this data in the form of a

result table. These result tables are called result-sets.

Syntax

The basic syntax of the SELECT statement is as follows −

SELECT column1, column2, columnN FROM table\_name;

Here, column1, column2... are the fields of a table whose values you want to fetch.

If you want to fetch all the fields available in the field, then you can use the

following syntax.

SELECT \* FROM table\_name;

mysql>select \* from customers;

+----+----------+-----+-----------+----------+

| ID | name | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 3001.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 3001.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | Pune | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 8 | Amal | 30 | Kochi | 30000.00 |

+----+----------+-----+-----------+----------+

The following code is an example, which would fetch the ID, Name and Salary fields

of the customers available in CUSTOMERS table.

mysql> select id,name,salary from customers;

+----+----------+----------+

| id | name | salary |

+----+----------+----------+

| 1 | Ramesh | 3001.00 |

| 2 | Khilan | 1500.00 |

| 3 | kaushik | 3001.00 |

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

| 8 | Amal | 30000.00 |

+----+----------+----------+

-------------

If you want to fetch all the fields of the CUSTOMERS

table, then you should use the following query.

mysql> SELECT \* FROM CUSTOMERS;

------------

SQL - WHERE Clause

The SQL WHERE clause is used to specify a condition while

fetching the data from a single table or by joining with

multiple tables. If the given condition is satisfied,

then only it returns a specific value from the table.

You should use the WHERE clause to filter the records

and fetching only the necessary records.

The WHERE clause is not only used in the SELECT

statement,

but it is also used

in the UPDATE, DELETE statement, etc.

Syntax

The basic syntax of the SELECT statement with the WHERE

clause is as shown below.

SELECT column1, column2,..., columnN FROM table\_name

WHERE <condition>

You can specify a condition using the comparison or

logical operators like >, <, =, LIKE, NOT, etc.

Fetch the ID, Name and Salary fields from the CUSTOMERS

table, where the salary is greater than 4000 −

mysql> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 4000;

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

| 8 | Amal | 30000.00 |

+----+----------+----------+

The following query is an example, which would fetch the

ID, Name and Salary fields from the CUSTOMERS table for

a customer with the name Hardik.

mysql> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE NAME = 'Hardik';

+----+--------+---------+

| ID | NAME | SALARY |

+----+--------+---------+

| 5 | Hardik | 8500.00 |

+----+--------+---------+

SQL - AND and OR Conjunctive Operators

The SQL AND & OR operators are used to combine multiple

conditions to narrow data in an SQL statement.

These two operators are called as the conjunctive

operators.

These operators provide a means to make multiple

comparisons with different operators in the same SQL

statement.

The AND Operator

The AND operator allows the existence of multiple

conditions in an SQL statement's WHERE clause.

Syntax

The basic syntax of the AND operator with a WHERE clause is as follows −

SELECT column1, column2, columnN FROM table\_name WHERE <condition1>

AND <condition2>...AND <conditionN>;

You can combine N number of conditions using the AND

operator. For an action to be taken by the SQL statement,

whether it be a transaction or a query, all conditions

separated by the AND must be TRUE.

----------------

Fetch the ID, Name and Salary fields from the CUSTOMERS

table, where the salary is greater than 3000 and the age

is less than 25 years −

mysql> SELECT ID, NAME, SALARY FROM

CUSTOMERS WHERE SALARY > 3000 AND age < 25;

+----+---------+----------+

| ID | NAME | SALARY |

+----+---------+----------+

| 3 | kaushik | 3001.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+---------+----------+

-----------------

The OR Operator

The OR operator is used to combine multiple conditions

in an SQL statement's WHERE clause.

Syntax

The basic syntax of the OR operator with a WHERE clause

is as follows −

SELECT column1, column2, columnN FROM table\_name

WHERE <condition1> OR <condition2>...OR <conditionN>

You can combine N number of conditions using the OR

operator. For an action to be taken by the SQL statement,

whether it be a transaction or query, the only any ONE

of the conditions separated by the OR must be TRUE.

fetch the ID, Name and Salary fields from the CUSTOMERS

table, where the salary is greater than 4000 OR the age

is less than 25 years.

mysql> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY >4000 OR age < 23;

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

mysql>SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY >4000 OR (age < 30 and age>25);

+----+----------+----------+

| ID | NAME | SALARY |

+----+----------+----------+

| 4 | Chaitali | 6500.00 |

| 5 | Hardik | 8500.00 |

| 6 | Komal | 4500.00 |

| 7 | Muffy | 10000.00 |

+----+----------+----------+

----------------

To clear mysql prompt screen

mysql> \! cls;

-------------------------

NOT Operator

mysql> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE not(SALARY >4000);

+----+---------+---------+

| ID | NAME | SALARY |

+----+---------+---------+

| 1 | Ramesh | 2000.00 |

| 2 | Khilan | 1500.00 |

| 3 | kaushik | 2000.00 |

+----+---------+---------+

----------------------

SQL - UPDATE Query

The SQL UPDATE Query is used to modify the existing

records in a table. You can use the WHERE clause

with the UPDATE query to update the selected rows,

otherwise all the rows would be affected.

mysql> select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Syntax

The basic syntax of the UPDATE query with a WHERE clause

is as follows −

UPDATE table\_name SET column1 = value1, column2 = value2

...., columnN = valueN WHERE [condition];

Update the ADDRESS for a customer whose ID number

is 6 in the table.

mysql> UPDATE CUSTOMERS SET ADDRESS = 'Kochi' WHERE ID = 6;

Query OK, 1 row affected (0.05 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | Kochi | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

------------------------------

SQL - DELETE Query

The SQL DELETE Query is used to delete the existing

records from a table.

You can use the WHERE clause with a DELETE query to delete

the selected rows, otherwise all the records would be

deleted.

Syntax

The basic syntax of the DELETE query with the WHERE clause

is as follows −

DELETE FROM table\_name WHERE [condition];

DELETE a customer, whose ID is 6.

mysql> DELETE FROM CUSTOMERS WHERE ID = 6;

Query OK, 1 row affected (0.04 sec)

mysql> select \* from customers;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

If you want to DELETE all the records from the

CUSTOMERS table, you do not need to use the WHERE

clause

----------------------------------

SQL - LIKE Clause

The SQL LIKE clause is used to compare a value to

similar values using wildcard operators. There are

two wildcards used in conjunction with the LIKE

operator.

The percent sign (%)

The underscore (\_)

The percent sign represents zero, one or multiple

characters. The underscore represents a single

number or character. These symbols can be used in

combinations.

Finds any values that start with 200.

Finds any values that have 200 in starting position.

mysql> SELECT \* FROM CUSTOMERS WHERE SALARY LIKE '200%';

+----+---------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+---------+-----+-----------+---------+

Finds any values that have 00 in the second and

third positions.

mysql> SELECT \* FROM CUSTOMERS WHERE SALARY LIKE '\_00%';

+----+---------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+---------+-----+-----------+----------+

3 rows in set (0.00 sec)

Finds any values that start with 2 and are at least 3 characters in length.

mysql> SELECT \* FROM CUSTOMERS WHERE SALARY LIKE '2\_%\_%';

+----+---------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+---------+-----+-----------+---------+

-------------------------------------

SQL - TOP, LIMIT or ROWNUM Clause

The SQL TOP clause is used to fetch a TOP N number

or X percent records from a table.

Note − All the databases do not support the TOP

clause. For example MySQL supports the LIMIT

clause to fetch limited number of records while

Oracle uses the ROWNUM command to fetch a limited

number of records.

Syntax

The basic syntax of the TOP clause with a SELECT

statement would be as follows.

For mysql:

SELECT col1,col2,... FROM table\_name LIMIT count;

If you are using MySQL server, then here is an

equivalent example −

list first 3 records

mysql> SELECT \* FROM CUSTOMERS LIMIT 3;

+----+---------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+---------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

+----+---------+-----+-----------+---------+

List 3 records from 3rd position

mysql> SELECT \* FROM customers LIMIT 3 OFFSET 2;

+----+----------+-----+---------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+---------+

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

+----+----------+-----+---------+---------+

-------------------------------

SQL - ORDER BY Clause

The SQL ORDER BY clause is used to sort the data in

ascending or descending order, based on one or more

columns. Some databases sort the query results in an

ascending order by default.

Syntax

The basic syntax of the ORDER BY clause is as

follows −

SELECT column-list FROM table\_name [WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

Sort the result in an ascending order by SALARY −

mysql> SELECT \* FROM CUSTOMERS ORDER BY SALARY;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

mysql> SELECT \* FROM CUSTOMERS ORDER BY SALARY,AGE;

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

--------------------------------

SQL - Group By

The SQL GROUP BY clause is used in collaboration

with the SELECT statement to arrange identical

data into groups. This GROUP BY clause follows

the WHERE clause in a SELECT statement and

precedes the ORDER BY clause.

Syntax

The basic syntax of a GROUP BY clause is shown

in the following code block. The GROUP BY clause

must follow the conditions in the WHERE clause

and must precede the ORDER BY clause if one is used.

SELECT column1, column2 FROM table\_name WHERE [ conditions ]

GROUP BY column1, column2 ORDER BY column1, column2

If you want to know the total amount of the salary

on each customer, then the GROUP BY query would

be as follows.

mysql> SELECT NAME, SUM(SALARY) FROM CUSTOMERS GROUP BY NAME;

mysql> create table employee(id int, name varchar(30), desi varchar(30), salary int, primary key(id));

Query OK, 0 rows affected (0.14 sec)

mysql> desc employee;

+--------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------+-------------+------+-----+---------+-------+

| id | int | NO | PRI | NULL | |

| name | varchar(30) | YES | | NULL | |

| desi | varchar(30) | YES | | NULL | |

| salary | int | YES | | NULL | |

+--------+-------------+------+-----+---------+-------+

4 rows in set (0.06 sec)

mysql> insert into employee values(100,'Alex','developer', 46000);

Query OK, 1 row affected (0.04 sec)

mysql> insert into employee values(100,'kumar','developer', 54000);

ERROR 1062 (23000): Duplicate entry '100' for key 'employee.PRIMARY'

mysql> insert into employee values(101,'kumar','developer', 54000);

Query OK, 1 row affected (0.04 sec)

mysql> insert into employee values(102,'Raj','designer', 34000);

Query OK, 1 row affected (0.00 sec)

mysql> insert into employee values(104,'Kamal','designer', 40000);

Query OK, 1 row affected (0.04 sec)

mysql> select \* from employee;

+-----+-------+-----------+--------+

| id | name | desi | salary |

+-----+-------+-----------+--------+

| 100 | Alex | developer | 46000 |

| 101 | kumar | developer | 54000 |

| 102 | Raj | designer | 34000 |

| 104 | Kamal | designer | 40000 |

+-----+-------+-----------+--------+

4 rows in set (0.00 sec)

mysql> select sum(salary) group by desi;

ERROR 1054 (42S22): Unknown column 'salary' in 'field list'

mysql> select sum(salary) from employee group by desi;

+-------------+

| sum(salary) |

+-------------+

| 100000 |

| 74000 |

+-------------+

2 rows in set (0.00 sec)

mysql>

----------------------

SQL - Distinct Keyword

The SQL DISTINCT keyword is used in conjunction

with the SELECT statement to eliminate all the

duplicate records and fetching only unique records.

There may be a situation when you have multiple

duplicate records in a table. While fetching such

records, it makes more sense to fetch only those

unique records instead of fetching duplicate

records.

Syntax

The basic syntax of DISTINCT keyword to eliminate

the duplicate records is as follows −

SELECT DISTINCT column1, column2,.....columnN FROM table\_name

WHERE [condition]

First,

mysql> select \* from customers;

+----+----------+-----+-----------+----------+

| ID | name | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 3001.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 3001.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | Kochi | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 8 | Amal | 30 | Kochi | 30000.00 |

| 9 | Ajith | 28 | Kochi | 30000.00 |

+----+----------+-----+-----------+----------+

First, let us see how the following SELECT query returns the duplicate salary records.

mysql> SELECT \* FROM CUSTOMERS ORDER BY SALARY;

+----+----------+-----+-----------+----------+

| ID | name | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 1 | Ramesh | 32 | Ahmedabad | 3001.00 |

| 3 | kaushik | 23 | Kota | 3001.00 |

| 6 | Komal | 22 | Kochi | 4500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 8 | Amal | 30 | Kochi | 30000.00 |

| 9 | Ajith | 28 | Kochi | 30000.00 |

+----+----------+-----+-----------+----------+

Now, let us use the DISTINCT keyword with the above SELECT query and then

see the result.

mysql> SELECT DISTINCT SALARY FROM CUSTOMERS ORDER BY SALARY;

+----------+

| SALARY |

+----------+

| 1500.00 |

| 3001.00 |

| 4500.00 |

| 6500.00 |

| 8500.00 |

| 10000.00 |

| 30000.00 |

+----------+

mysql> SELECT DISTINCT ADDRESS,SALARY FROM CUSTOMERS ORDER BY SALARY;

+-----------+----------+

| ADDRESS | SALARY |

+-----------+----------+

| Delhi | 1500.00 |

| Ahmedabad | 3001.00 |

| Kota | 3001.00 |

| Kochi | 4500.00 |

| Mumbai | 6500.00 |

| Bhopal | 8500.00 |

| Indore | 10000.00 |

| Kochi | 30000.00 |

+-----------+----------+

mysql> update employee set salary =46000 where id=101;

Query OK, 1 row affected (0.04 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> select \* from employee;

+-----+-------+-----------+--------+

| id | name | desi | salary |

+-----+-------+-----------+--------+

| 100 | Alex | developer | 46000 |

| 101 | kumar | developer | 46000 |

| 102 | Raj | designer | 34000 |

| 104 | Kamal | designer | 40000 |

+-----+-------+-----------+--------+

4 rows in set (0.00 sec)

mysql> select distinct desi from employee;

+-----------+

| desi |

+-----------+

| developer |

| designer |

+-----------+

2 rows in set (0.00 sec)

mysql> select distinct desi,salary from employee;

+-----------+--------+

| desi | salary |

+-----------+--------+

| developer | 46000 |

| designer | 34000 |

| designer | 40000 |

+-----------+--------+

3 rows in set (0.00 sec)

mysql>

-------------------------

SQL - SORTING Results

The SQL ORDER BY clause is used to sort the data

in ascending or descending order, based on one

or more columns. Some databases sort the query

results in an ascending order by default.

Syntax

The basic syntax of the ORDER BY clause which

would be used to sort the result in an ascending

or descending order is as follows −

SELECT column-list FROM table\_name [WHERE condition] [ORDER BY column1,

column2, .. columnN] [ASC | DESC];

First List all,

mysql> SELECT \* FROM CUSTOMERS;

+----+----------+-----+-----------+----------+

| ID | name | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 3001.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 3001.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | Kochi | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 8 | Amal | 30 | Kochi | 30000.00 |

| 9 | Ajith | 28 | Kochi | 30000.00 |

+----+----------+-----+-----------+----------+

sort the result in an ascending order by SALARY.

mysql> SELECT \* FROM CUSTOMERS ORDER BY SALARY;

+----+----------+-----+-----------+----------+

| ID | name | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 1 | Ramesh | 32 | Ahmedabad | 3001.00 |

| 3 | kaushik | 23 | Kota | 3001.00 |

| 6 | Komal | 22 | Kochi | 4500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 8 | Amal | 30 | Kochi | 30000.00 |

| 9 | Ajith | 28 | Kochi | 30000.00 |

+----+----------+-----+-----------+----------+

sort the result in an ascending order by NAME and SALARY.

mysql> SELECT \* FROM CUSTOMERS ORDER BY NAME, SALARY;

+----+----------+-----+-----------+----------+

| ID | name | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 9 | Ajith | 28 | Kochi | 30000.00 |

| 8 | Amal | 30 | Kochi | 30000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 3 | kaushik | 23 | Kota | 3001.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 6 | Komal | 22 | Kochi | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 1 | Ramesh | 32 | Ahmedabad | 3001.00 |

+----+----------+-----+-----------+----------+

sort the result in a descending order by NAME.

mysql>SELECT \* FROM CUSTOMERS ORDER BY NAME DESC;

+----+----------+-----+-----------+----------+

| ID | name | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 3001.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

| 6 | Komal | 22 | Kochi | 4500.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 3001.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 8 | Amal | 30 | Kochi | 30000.00 |

| 9 | Ajith | 28 | Kochi | 30000.00 |

+----+----------+-----+-----------+----------+

----------------------------

SQL - Using Joins

The SQL Joins clause is used to combine records from two

or more tables in a database. A JOIN is a means for

combining fields from two tables by using values common

to each.

CUSTOMERS Table

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Table 2 − ORDERS Table

mysql>CREATE TABLE ORDERS(oid int,date date ,customer\_id int ,amount int ,primary key(oid));

mysql>insert into orders(oid ,date,customer\_id,amount)values (102,'2009-10-08 00:00:00',3,3000);

mysql>insert into orders(oid ,date,customer\_id,amount)values(100,' 2009-10-08 00:00:00 ',3,1500);

mysql>insert into orders(oid ,date,customer\_id,amount)values(101,'2009-11-20 00:00:00',2,1560);

mysql>insert into orders(oid ,date,customer\_id,amount)values(103,'2008-05-20 00:00:00 ',4,2060);

mysql> select \* from orders;

+-----+------------+-------------+--------+

| oid | date | customer\_id | amount |

+-----+------------+-------------+--------+

| 100 | 2009-10-08 | 3 | 1500 |

| 101 | 2009-11-20 | 2 | 1560 |

| 102 | 2009-10-08 | 3 | 3000 |

| 103 | 2008-05-20 | 4 | 2060 |

+-----+------------+-------------+--------+

Several operators can be used to join tables, such

as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and

NOT, they can all be used to join tables.

There are different types of joins available in SQL

SQL - INNER JOINS

The most important and frequently used of the joins

is the INNER JOIN. They are also referred to as an

EQUIJOIN.

The INNER JOIN creates a new result table by

combining column values of two tables (table1

and table2) based upon the join-predicate. The

query compares each row of table1 with each row

of table2 to find all pairs of rows which satisfy

the join-predicate. When the join-predicate

is satisfied, column values for each matched pair

of rows of A and B are combined into a result row.

Syntax

The basic syntax of the INNER JOIN is as follows.

SELECT table1.column1, table2.column2... FROM table1 INNER JOIN table2

ON table1.common\_field = table2.common\_field;

mysql> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS

INNER JOIN ORDERS ON

CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

+----+----------+--------+------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+------------+

| 3 | kaushik | 1500 | 2009-10-08 |

| 2 | Khilan | 1560 | 2009-11-20 |

| 3 | kaushik | 3000 | 2009-10-08 |

| 4 | Chaitali | 2060 | 2008-05-20 |

+----+----------+--------+------------+

----------------------------

SQL - LEFT JOINS

The SQL LEFT JOIN returns all rows from the left

table, even if there are no matches in the right

table. This means that if the ON clause matches

0 (zero) records in the right table; the join

will still return a row in the result, but with

NULL in each column from the right table.

Syntax

The basic syntax of a LEFT JOIN is as follows.

SELECT table1.column1, table2.column2... FROM table1 LEFT JOIN table2

ON table1.common\_field = table2.common\_field;

mysql> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS LEFT JOIN ORDERS ON

CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

+----+----------+--------+------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 |

| 3 | kaushik | 3000 | 2009-10-08 |

| 3 | kaushik | 1500 | 2009-10-08 |

| 4 | Chaitali | 2060 | 2008-05-20 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

| 8 | Amal | NULL | NULL |

| 9 | Ajith | NULL | NULL |

+----+----------+--------+------------+

----------------------------------------

SQL - RIGHT JOINS

The SQL RIGHT JOIN returns all rows from the right

table, even if there are no matches in the left

table. This means that if the ON clause matches

0 (zero) records in the left table; the join will

still return a row in the result, but with NULL

in each column from the left table.

This means that a right join returns all the values

from the right table, plus matched values from the

left table or NULL in case of no matching join

predicate.

Syntax

The basic syntax of a RIGHT JOIN is as follow.

SELECT table1.column1, table2.column2... FROM table1 RIGHT JOIN table2

ON table1.common\_field = table2.common\_field;

mysql> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS RIGHT JOIN ORDERS ON

CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

+------+----------+--------+------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+------------+

| 3 | kaushik | 1500 | 2009-10-08 |

| 2 | Khilan | 1560 | 2009-11-20 |

| 3 | kaushik | 3000 | 2009-10-08 |

| 4 | Chaitali | 2060 | 2008-05-20 |

+------+----------+--------+------------+

--------------------------------------------

MySQL CROSS JOIN

The SQL CROSS JOIN combines the results of both

left and right outer joins.

The joined table will contain all records from

both the tables and fill in

NULLs for missing matches on either side.

Syntax

The basic syntax of a CROSS JOIN(FULL JOIN) is as

follows −

The CROSS JOIN keyword returns all matching

records from both tables whether the other table

matches or not. So, if there are rows in

"Customers" that do not have matches

in "Orders", or if there are rows in "Orders" that

do not have matches in "Customers", those rows

will be listed as well.

syntax:

SELECT table1.column1, table2.column2... FROM table1 CROSS

JOIN table2 ON table1.common\_field = table2.common\_field;

mysql> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS CROSS JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

+----+----------+--------+------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+------------+

| 3 | kaushik | 1500 | 2009-10-08 |

| 2 | Khilan | 1560 | 2009-11-20 |

| 3 | kaushik | 3000 | 2009-10-08 |

| 4 | Chaitali | 2060 | 2008-05-20 |

+----+----------+--------+------------+

----------------------------------------------------

MySQL IN Condition

The MySQL IN condition is used to reduce the use

of multiple OR conditions in a SELECT, INSERT,

UPDATE and DELETE statement.

Syntax:

expression IN (value1, value2, .... value\_n);

expression: It specifies a value to test.

value1, value2, ... or value\_n: These are the

values to test against expression. If any of

these values matches expression, then the IN

condition will evaluate to true. This is a quick method to test if any one of the values matches expression.

mysql>select \* from customers where name in('Ramesh','Khilan','Hardik');

+----+--------+-----+-----------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+--------+-----+-----------+---------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

+----+--------+-----+-----------+---------+

List details of ordered customers

mysql>select \* from customers where id in(select customer\_id from orders);

+----+----------+-----+---------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+---------+

| 3 | kaushik | 23 | Kota | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

+----+----------+-----+---------+---------+

mysql> select \* from customers where id in(select customer\_id from orders where amount>2000);

+----+----------+-----+---------+---------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+---------+---------+

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

+----+----------+-----+---------+---------+

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Python MySQL

Python can be used in database applications.

One of the most popular databases is MySQL.

Install MySQL Driver

Python needs a MySQL driver to access the MySQL database.

We will use the driver "MySQL Connector".

We recommend that you use PIP to install "MySQL Connector".

Download and install "MySQL Connector":

python -m pip install mysql-connector-python

Oepn anaconda prompt:

(base)C:\Users\BAIJU>python -m pip install mysql-connector-python

Test MySQL Connector

To test if the installation was successful, or if you already have "MySQL Connector" installed,

create a Python page with the following content:

------------------------

import mysql.connector

print("connected")

--------------------------

Create Connection:

Start by creating a connection to the database.

Use the username and password from your MySQL database:

Syntax:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="yourusername",

password="yourpassword"

)

#print(mydb)

------------

Connecting with username and password

---------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root"

)

print(mydb)

output:

<mysql.connector.connection\_cext.CMySQLConnection object at 0x00000181026B2280>

---------------

Python MySQL Create Database

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

create a database named "mydatabase":

--------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root"

)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")

print("Database created..")

-------------------------------

Check if Database Exists

You can check if a database exist by listing all databases in your system by

using the "SHOW DATABASES" statement:

------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root"

)

mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

for x in mycursor:

print(x)

--------------------------

Try connecting to the database "mydatabase":

-----------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

print("connected...")

----------------------------

Python MySQL Create Table

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

Create a table named "customers":

------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (name VARCHAR(30), address VARCHAR(50))")

print("done")

If the above code was executed with no errors, you have now successfully created a table.

-----------------------

Check if Table Exists

You can check if a table exist by listing all tables in your database with the

"SHOW TABLES" statement:

Return a list of your system's databases:

--------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("SHOW TABLES")

for x in mycursor:

print(x)

--------------------------------

Python MySQL Insert Into Table

Insert Into Table

To fill a table in MySQL, use the "INSERT INTO" statement.

Insert a record in the "customers" table:

----------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

val = ("John", "Highway 21")

mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record inserted.")

----------------------------------

Important!: Notice the statement: mydb.commit(). It is

required to make the changes, otherwise no changes are

made to the table.

-----------------------

Insert Multiple Rows

To insert multiple rows into a table, use the executemany() method.

The second parameter of the executemany() method is a list of tuples,

containing the data you want to insert:

Fill the "customers" table with data:

--------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

val = [

('Peter', 'Lowstreet 4'),

('Amy', 'Apple st 652'),

('Hannah', 'Mountain 21'),

('Michael', 'Valley 345'),

('Sandy', 'Ocean blvd 2'),

('Betty', 'Green Grass 1'),

('Richard', 'Sky st 331'),

('Susan', 'One way 98'),

('Vicky', 'Yellow Garden 2'),

('Ben', 'Park Lane 38'),

('William', 'Central st 954'),

('Chuck', 'Main Road 989'),

('Viola', 'Sideway 1633')

]

mycursor.executemany(sql, val)

mydb.commit()

print(mycursor.rowcount, "was inserted.")

--------------------

Python MySQL Select From

Select From a Table

To select from a table in MySQL, use the "SELECT" statement:

Select all records from the "customers" table, and display the result:

---------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM customers")

myresult = mycursor.fetchall()

for x in myresult:

print(x)

----------------------------

Note: We use the fetchall() method, which fetches all rows from the last executed statement.

-------------------------

Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed

by the column name(s):

Select only the name and address columns:

---------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("SELECT name, address FROM customers")

myresult = mycursor.fetchall()

for x in myresult:

print(x)

------------------------------

Using the fetchone() Method

If you are only interested in one row, you can use the fetchone() method.

The fetchone() method will return the first row of the result:

Fetch only one row:

---------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM customers")

myresult = mycursor.fetchone()

print(myresult)

--------------------------

Python MySQL Where

Select With a Filter

When selecting records from a table, you can filter the selection by using

the "WHERE" statement:

Select record(s) where the address is "Park Lane 38": result:

-----------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT \* FROM customers WHERE address ='Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

-------------------

Prevent SQL Injection

When query values are provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to

destroy or misuse your database.

The mysql.connector module has methods to escape query values:

Escape query values by using the placholder %s method:

----------------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT \* FROM customers WHERE address = %s"

adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

----------------------

Python MySQL Order By

Sort the Result

Use the ORDER BY statement to sort the result in ascending or descending order.

The ORDER BY keyword sorts the result ascending by default. To sort the result

in descending order, use the DESC keyword.

Sort the result alphabetically by name: result:

----------------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT \* FROM customers ORDER BY name"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

------------------------------

ORDER BY DESC

Use the DESC keyword to sort the result in a descending order.

Sort the result reverse alphabetically by name:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT \* FROM customers ORDER BY name DESC"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

---------------------

Python MySQL Delete From By Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

Delete any record where the address is "Mountain 21":

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = 'Mountain 21'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")

----------------------

Prevent SQL Injection

It is considered a good practice to escape the values of any query, also

in delete statements.

This is to prevent SQL injections, which is a common web hacking technique

to destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the

delete statement:

Escape values by using the placeholder %s method:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "DELETE FROM customers WHERE address = %s"

adr = ("Yellow Garden 2", )

mycursor.execute(sql, adr)

mydb.commit()

print(mycursor.rowcount, "record(s) deleted")

--------------------------------

Python MySQL Update Table

Update Table

You can update existing records in a table by using the "UPDATE" statement:

Overwrite the address column from "Valley 345" to "Canyon 123":

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) affected")

--------------------------------

Prevent SQL Injection in update

It is considered a good practice to escape the values of any query, also in

update statements.

This is to prevent SQL injections, which is a common web hacking technique to

destroy or misuse your database.

The mysql.connector module uses the placeholder %s to escape values in the delete

statement:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "UPDATE customers SET address = %s WHERE address = %s"

val = ("Valley 345", "Canyon 123")

mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record(s) affected")

-----------------------

Python MySQL Limit

Limit the Result

You can limit the number of records returned from the query, by using the

"LIMIT" statement:

Select the 5 first records in the "customers" table:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM customers LIMIT 5")

myresult = mycursor.fetchall()

for x in myresult:

print(x)

------------------------

Start From Another Position

If you want to return five records, starting from the third record, you can use the

"OFFSET" keyword:

Start from position 3, and return 5 records:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

mycursor.execute("SELECT \* FROM customers LIMIT 5 OFFSET 2")

myresult = mycursor.fetchall()

for x in myresult:

print(x)

----------------------------------------

Python MySQL Join:

Join Two or More Tables

You can combine rows from two or more tables, based on a related column

between them, by using a JOIN statement.

Consider you have a "users" table and a "products" table:

users:

mysql>create table users (id int, name varchar(30),fav int);

mysql>insert into users(id,name,fav)values(1,'john',154);

mysql>insert into users(id,name,fav)values(2,'peter',154);

mysql>insert into users(id,name,fav)values(3,'amy',155);

products:

mysql>create table products(id int , name varchar(30));

mysql>insert into products(id,name) values(154,'Chocolate Heaven');

mysql>insert into products(id,name) values(155,'Tasty Lemons');

mysql>insert into products(id,name) values(156,'Vanilla Dreams');

mysql>select \* from users;

mysql>select \* from products

----------------------------

Join users and products to see the name of the users favorite product:

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT \

users.name , \

products.name \

FROM users \

INNER JOIN products ON users.fav = products.id"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

---------------------------

LEFT JOIN

In the example above, Hannah, and Michael were excluded from the result,

that is because INNER JOIN only shows the records where there is a match.

If you want to show all users, even if they do not have a favorite product,

use the LEFT JOIN statement:

----------------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT \

users.name , \

products.name \

FROM users \

LEFT JOIN products ON users.fav = products.id"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

--------------------------------

RIGHT JOIN

If you want to return all products, and the users who have them as their favorite,

even if no user have them as their favorite, use the RIGHT JOIN statement:

------------------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "SELECT \

users.name , \

products.name \

FROM users \

RIGHT JOIN products ON users.fav = products.id"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:

print(x)

--------------

Python MySQL Drop Table:

Delete a Table

You can delete an existing table by using the "DROP TABLE" statement:

Delete the table "abc":

---------------------------------

import mysql.connector

mydb = mysql.connector.connect(

host="localhost",

user="root",

password="root",

database="mydatabase"

)

mycursor = mydb.cursor()

sql = "DROP TABLE abc"

mycursor.execute(sql)