

```

from tkinter import *
import tkinter.messagebox as tkMessageBox

import sqlite3
import tkinter.ttk as ttk

root = Tk()

root.title("Simple Inventory System")

width = 1024
height = 520
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
root.geometry("%dx%d+%d+%d" % (width, height, x, y))

root.resizable(0, 0)
root.config(bg="#6666ff")

#
=====VARIABLES=====
=====
USERNAME = StringVar()
PASSWORD = StringVar()
PRODUCT_NAME = StringVar()
PRODUCT_PRICE = IntVar()
PRODUCT_QTY = IntVar()
SEARCH = StringVar()

```

```

#
=====METHODS=====
=====

def Database():
    global conn, cursor
    conn = sqlite3.connect("pythontut.db")

    cursor = conn.cursor()

    cursor.execute(

        "CREATE TABLE IF NOT EXISTS `admin` (admin_id INTEGER
PRIMARY KEY AUTOINCREMENT NOT NULL, username TEXT, password
TEXT)")

    cursor.execute(

        "CREATE TABLE IF NOT EXISTS `product` (product_id
INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, product_name TEXT,
product_qty TEXT, product_price TEXT)")

    cursor.execute("SELECT * FROM `admin` WHERE `username` =
'admin' AND `password` = 'admin'")

    if cursor.fetchone() is None:

        cursor.execute("INSERT INTO `admin` (username,
password) VALUES('admin', 'admin'")

        conn.commit()

def Exit():
    result = tkMessageBox.askquestion('Simple Inventory
System', 'Are you sure you want to exit?', icon="warning")

    if result == 'yes':
        root.destroy()
        exit()

def Exit2():
    result = tkMessageBox.askquestion('Simple Inventory
System', 'Are you sure you want to exit?', icon="warning")

    if result == 'yes':

```

Home.destroy()
exit()
def ShowLoginForm():
global loginform
loginform = Toplevel()
loginform.title("Simple Inventory System/Account Login")
width = 600
height = 500
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
loginform.resizable(0, 0)
loginform.geometry("%dx%d+%d+%d" % (width, height, x, y))
LoginForm()
def LoginForm():
global lbl_result
TopLoginForm = Frame(loginform, width=600, height=100, bd=1, relief=SOLID)
TopLoginForm.pack(side=TOP, pady=20)
lbl_text = Label(TopLoginForm, text="Administrator Login", font=('arial', 18), width=600)
lbl_text.pack(fill=X)
MidLoginForm = Frame(loginform, width=600)
MidLoginForm.pack(side=TOP, pady=50)
lbl_username = Label(MidLoginForm, text="Username:", font=('arial', 25), bd=18)
lbl_username.grid(row=0)
lbl_password = Label(MidLoginForm, text="Password:", font=('arial', 25), bd=18)
lbl_password.grid(row=1)
lbl_result = Label(MidLoginForm, text="", font=('arial', 18))
lbl_result.grid(row=3, columnspan=2)

```

    username = Entry(MidLoginForm, textvariable=USERNAME,
font=('arial', 25), width=15)

    username.grid(row=0, column=1)

    password = Entry(MidLoginForm, textvariable=PASSWORD,
font=('arial', 25), width=15, show="*")

    password.grid(row=1, column=1)

    btn_login = Button(MidLoginForm, text="Login",
font=('arial', 18), width=30, command=Login)

    btn_login.grid(row=2, columnspan=2, pady=20)

    btn_login.bind('<Return>', Login)


def Home():
    global Home
    Home = Tk()

    Home.title("Simple Inventory System/Home")

    width = 1024
    height = 520

    screen_width = Home.winfo_screenwidth()
    screen_height = Home.winfo_screenheight()

    x = (screen_width / 2) - (width / 2)
    y = (screen_height / 2) - (height / 2)

    Home.geometry("%dx%d+%d+%d" % (width, height, x, y))

    Home.resizable(0, 0)

    Title = Frame(Home, bd=1, relief=SOLID)

    Title.pack(pady=10)

    lbl_display = Label(Title, text="Simple Inventory System",
font=('arial', 45))

    lbl_display.pack()

    menubar = Menu(Home)

    filemenu = Menu(menubar, tearoff=0)

    filemenu2 = Menu(menubar, tearoff=0)

    filemenu.add_command(label="Logout", command=Logout)

    filemenu.add_command(label="Exit", command=Exit2)

    filemenu2.add_command(label="Add new", command=ShowAddNew)

    filemenu2.add_command(label="View", command=ShowView)

    menubar.add_cascade(label="Account", menu=filemenu)

    menubar.add_cascade(label="Inventory", menu=filemenu2)

```

Home.config(menu=menubar)
Home.config(bg="#6666ff")
def ShowAddNew():
global addnewform
addnewform = Toplevel()
addnewform.title("Simple Inventory System/Add new")
width = 600
height = 500
screen_width = Home.winfo_screenwidth()
screen_height = Home.winfo_screenheight()
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
addnewform.geometry("%dx%d+%d+%d" % (width, height, x, y))
addnewform.resizable(0, 0)
AddNewForm()
def AddNewForm():
TopAddNew = Frame(addnewform, width=600, height=100, bd=1, relief=SOLID)
TopAddNew.pack(side=TOP, pady=20)
lbl_text = Label(TopAddNew, text="Add New Product", font=('arial', 18), width=600)
lbl_text.pack(fill=X)
MidAddNew = Frame(addnewform, width=600)
MidAddNew.pack(side=TOP, pady=50)
lbl_productname = Label(MidAddNew, text="Product Name:", font=('arial', 25), bd=10)
lbl_productname.grid(row=0, sticky=W)
lbl_qty = Label(MidAddNew, text="Product Quantity:", font=('arial', 25), bd=10)
lbl_qty.grid(row=1, sticky=W)
lbl_price = Label(MidAddNew, text="Product Price:", font=('arial', 25), bd=10)
lbl_price.grid(row=2, sticky=W)
productname = Entry(MidAddNew, textvariable=PRODUCT_NAME, font=('arial', 25), width=15)
productname.grid(row=0, column=1)

```

    productqty = Entry(MidAddNew, textvariable=PRODUCT_QTY,
font=('arial', 25), width=15)

    productqty.grid(row=1, column=1)

    productprice = Entry(MidAddNew,
textvariable=PRODUCT_PRICE, font=('arial', 25), width=15)

    productprice.grid(row=2, column=1)

    btn_add = Button(MidAddNew, text="Save", font=('arial',
18), width=30, bg="#009ACD", command=AddNew)

    btn_add.grid(row=3, columnspan=2, pady=20)


def AddNew():
    Database()

    cursor.execute("INSERT INTO `product` (product_name,
product_qty, product_price) VALUES(?, ?, ?)",

                    (str(PRODUCT_NAME.get()),
int(PRODUCT_QTY.get()), int(PRODUCT_PRICE.get())) )

    conn.commit()

    PRODUCT_NAME.set("")
    PRODUCT_PRICE.set("")
    PRODUCT_QTY.set("")

    cursor.close()
    conn.close()


def ViewForm():
    global tree

    TopViewForm = Frame(viewform, width=600, bd=1,
relief=SOLID)

    TopViewForm.pack(side=TOP, fill=X)

    LeftViewForm = Frame(viewform, width=600)

    LeftViewForm.pack(side=LEFT, fill=Y)

    MidViewForm = Frame(viewform, width=600)

    MidViewForm.pack(side=RIGHT)

    lbl_text = Label(TopViewForm, text="View Products",
font=('arial', 18), width=600)

    lbl_text.pack(fill=X)

    lbl_txtsearch = Label(LeftViewForm, text="Search",
font=('arial', 15))

    lbl_txtsearch.pack(side=TOP, anchor=W)

    search = Entry(LeftViewForm, textvariable=SEARCH,
font=('arial', 15), width=10)

```

```

search.pack(side=TOP, padx=10, fill=X)

btn_search = Button(LeftViewForm, text="Search",
command=Search)

btn_search.pack(side=TOP, padx=10, pady=10, fill=X)

btn_reset = Button(LeftViewForm, text="Reset",
command=Reset)

btn_reset.pack(side=TOP, padx=10, pady=10, fill=X)

btn_delete = Button(LeftViewForm, text="Delete",
command=Delete)

btn_delete.pack(side=TOP, padx=10, pady=10, fill=X)

scrollbarx = Scrollbar(MidViewForm, orient=HORIZONTAL)

scrollbary = Scrollbar(MidViewForm, orient=VERTICAL)

tree = ttk.Treeview(MidViewForm, columns=("ProductID",
"Product Name", "Product Qty", "Product Price"),

selectmode="extended", height=100,
yscrollcommand=scrollbary.set, xscrollcommand=scrollbarx.set)

scrollbary.config(command=tree.yview)
scrollbary.pack(side=RIGHT, fill=Y)
scrollbarx.config(command=tree.xview)
scrollbarx.pack(side=BOTTOM, fill=X)
tree.heading('ProductID', text="ProductID", anchor=W)
tree.heading('Product Name', text="Product Name",
anchor=W)
tree.heading('Product Qty', text="Product Qty", anchor=W)
tree.heading('Product Price', text="Product Price",
anchor=W)
tree.column('#0', stretch=NO, minwidth=0, width=0)
tree.column('#1', stretch=NO, minwidth=0, width=0)
tree.column('#2', stretch=NO, minwidth=0, width=200)
tree.column('#3', stretch=NO, minwidth=0, width=120)
tree.column('#4', stretch=NO, minwidth=0, width=120)

tree.pack()
DisplayData()

```

```

def DisplayData():
    Database()
    cursor.execute("SELECT * FROM `product`")
    fetch = cursor.fetchall()
    for data in fetch:
        tree.insert('', 'end', values=(data))
    cursor.close()
    conn.close()

def Search():
    if SEARCH.get() != "":
        tree.delete(*tree.get_children())
        Database()
        cursor.execute("SELECT * FROM `product` WHERE `product_name` LIKE ?", ('%' + str(SEARCH.get()) + '%',))
        fetch = cursor.fetchall()
        for data in fetch:
            tree.insert('', 'end', values=(data))
        cursor.close()
        conn.close()

def Reset():
    tree.delete(*tree.get_children())
    DisplayData()
    SEARCH.set("")

def Delete():
    if not tree.selection():
        print("ERROR")
    else:
        result = tkMessageBox.askquestion('Simple Inventory System', 'Are you sure you want to delete this record?',
                                          icon="warning")

        if result == 'yes':
            curItem = tree.focus()
            contents = (tree.item(curItem))
            selecteditem = contents['values']
            tree.delete(curItem)

```


Database()
cursor.execute("DELETE FROM `product` WHERE `product_id` = %d" % selecteditem[0])
conn.commit()
cursor.close()
conn.close()
def ShowView():
global viewform
viewform = Toplevel()
viewform.title("Simple Inventory System/View Product")
width = 600
height = 400
screen_width = Home.winfo_screenwidth()
screen_height = Home.winfo_screenheight()
x = (screen_width / 2) - (width / 2)
y = (screen_height / 2) - (height / 2)
viewform.geometry("%dx%d+%d+%d" % (width, height, x, y))
viewform.resizable(0, 0)
ViewForm()
def Logout():
result = tkMessageBox.askquestion('Simple Inventory System', 'Are you sure you want to logout?', icon="warning")
if result == 'yes':
admin_id = ""
root.deiconify()
Home.destroy()
def Login(event=None):
global admin_id
Database()
if USERNAME.get == "" or PASSWORD.get() == "":
lbl_result.config(text="Please complete the required field!", fg="red")
else:

```

        cursor.execute("SELECT * FROM `admin` WHERE `username`
= ? AND `password` = ?",
                        (USERNAME.get(), PASSWORD.get()))

        if cursor.fetchone() is not None:

            cursor.execute("SELECT * FROM `admin` WHERE
`username` = ? AND `password` = ?",
                            (USERNAME.get(), PASSWORD.get()))

            data = cursor.fetchone()
            admin_id = data[0]
            USERNAME.set("")
            PASSWORD.set("")
            lbl_result.config(text="")
            ShowHome()

        else:
            lbl_result.config(text="Invalid username or
password", fg="red")
            USERNAME.set("")
            PASSWORD.set("")

    cursor.close()
    conn.close()

def ShowHome():
    root.withdraw()
    Home()
    loginform.destroy()

# =====MENUBAR
WIDGETS=====
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="Account", command=ShowLoginForm)
filemenu.add_command(label="Exit", command=Exit)
menubar.add_cascade(label="File", menu=filemenu)
root.config(menu=menubar)

#
=====FRAME=====
Title = Frame(root, bd=1, relief=SOLID)

```

```

Title.pack(pady=10)

# -----LABEL
WIDGET-----
lbl_display = Label(Title, text="Simple Inventory System",
font=('arial', 45))
lbl_display.pack()

#
-----INITIALIZATION-----

if __name__ == '__main__':
    root.mainloop()

```

code meaning. The coloured one is for changed tha
Import functions
Import function for ahowing message box (asking yes / no ...)
this is inbuilt database in python
initial function
creating tkinter window
root is object for the funcion . The tkinter window is made in starting.
The windows title parameter is given here.
make varaiaable width and give it a value= 1024 pixels
make variable height & give it a value = 520;
screen _width is a new variable which has been made.
screen _height is a new variable which has been made.
x is a variable, which has been given a formula.
similarly for height
(root ki geometry) - this is syntax for geometry - here we specify the complete width, height, x& y- e.i the starting xy point from where we make the window.
refer resizable function for window . Here we are fixing the size. It cannot change.
we have given a colour to background.
define data types to variables

Define functions:
we are making our own function called database
make these two global variables - conn & cursor
we define conn as a function which connects sqlite database called python tut.db
cursor - is a variable which is - connection ka cursor function. (causing execution of sql statements this is a default command for executing sql.
what ever we write for cursor.execute- it does the job.
this creates table in database; table name- is admin
create table - product
select the row- from admin, where user name and pass word is admin;
if there is no row in data base i.e. fetchone (), then insert user name and password
insert into admin
confirm (commit) the changes- we have to commit every time after insert command. Now it is saved. Commit - means saved.
define exit function
result is a variable, which creates a message box and in this box, we ask you want to exit
use destroy method, to close the root widget
define exit2 function

close home
showloginform function is made
login form is the top level window, in side we can have more windows
like we made root winddow, a new winfdoew has been made.
simila stapeas earlier.
call - loginForm function
define - loginForm function
define variable-
make a frame an nameir ass - Top LoginForm. Check the properties of frame .
.pack - means what ever we put in the window, it will automatically adjust in the window size. Chech the [properties for .pack)
his lable is called- lbl_text and a text is given - administrator;;
make anothe frman
make a label
.grid- we can arrange n grid
aanother lable

this places the munubar on home window
is backgrpund colour.

```
define show AddNew form function
```

call add new form function

define AddNew form function
we give layout to the box / label;
refer- geometry work sheet

added button
define AddNew function
nw added in database
value(?,?) - refer sqlite
these are sqlite commands to get the values
.set function is called.
define view form function

[illegible]

define display data function

define search function

define reset function

define delete function

cursoe ko execute karein, select everything from admin, where user name and password is given;
user name ko le lo; password ko le lo;
if fetch one function is not zero, then execute from admin;
data is a variable, which is from fetch one function
home call karein;
otherwise-
labl esult - invalid usr id;
user name- set karein - blank
curson - close karien;
connection to database to close karein;
define show home function
rook ko withdraw karen;
home ko call;
loingform ko destroy;
menu bar is a variable for menu in root;
refer menu bar worksheet
make a frame

label display
if main, then call root main loop function

[illegible]

[illegible]

In this article, database connection with the python program is discussed. Connecting a program with a database is considered a tough task in any programming language. It is used to connect the front-end of your application with the back-end database. Python with its native builtin modules made this thing easy too.		
This needs the basic understanding of SQL.		
Here, we are going to connect SQLite with Python. Python has a native library for SQLite. Let us explain how it works.		
1. To use SQLite, we must import sqlite3.		
2. Then create a connection using connect() method and pass the name of the database you want to access if there is a file with that name, it will open that file. Otherwise, Python will create a file with the given name.		
3. After this, a cursor object is called to be capable to send commands to the SQL. Cursor is a control structure used to traverse and fetch the records of the database. Cursor has a major role in working with Python. All the commands will be executed using cursor object only.		
4. To create a table in the database, create an object and write the SQL command in it with being commented. Example:- sql_comm = "SQL statement"		
5. And executing the command is very easy. Call the cursor method execute and pass the name of the sql command as a parameter in it. Save a number of commands as the sql_comm and execute them. After you perform all your activities, save the changes in the file by committing those changes and then lose the connection.		
# Python code to demonstrate table creation and		
# insertions with SQL		
# importing module		
import sqlite3		
# connecting to the database		
connection = sqlite3.connect("myTable.db")		
# cursor		
crsr = connection.cursor()		
# SQL command to create a table in the database		
sql_command = """CREATE TABLE emp (
staff_number INTEGER PRIMARY KEY,		

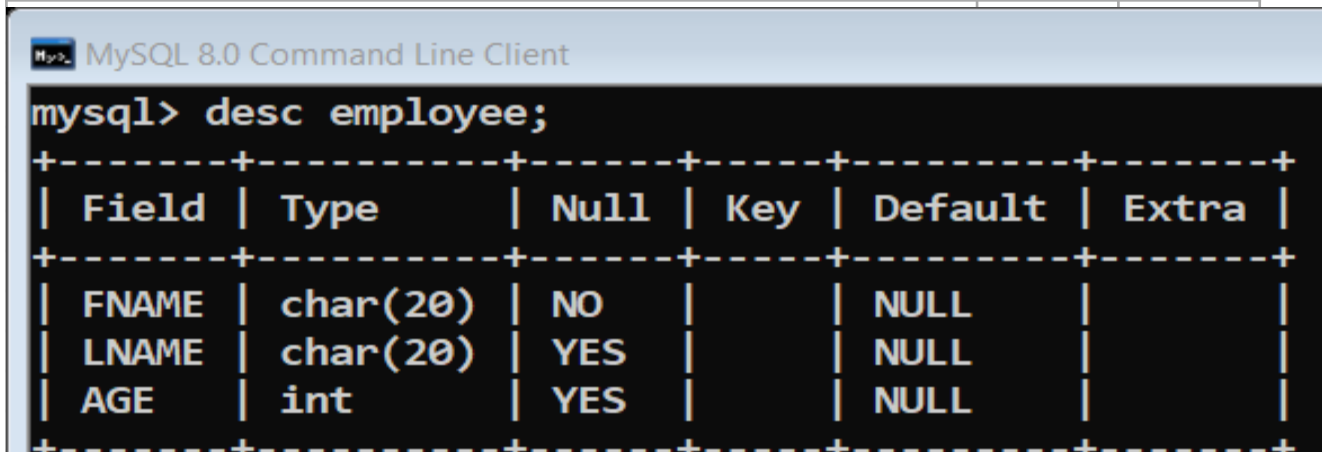
fname VARCHAR(20),		
lname VARCHAR(30),		
gender CHAR(1),		
joining DATE);""		
# execute the statement		
crsr.execute(sql_command)		
# SQL command to insert the data in the table		
sql_command = ""INSERT INTO emp VALUES (23, "Rishabh", "Bansal", "M", "2014-03-28");""		
crsr.execute(sql_command)		
# another SQL command to insert the data in the table		
sql_command = ""INSERT INTO emp VALUES (1, "Bill", "Gates", "M", "1980-10-28");""		
crsr.execute(sql_command)		
# To save the changes in the files. Never skip this.		
# If we skip this, nothing will be saved in the database.		
connection.commit()		
# close the connection		
connection.close()		
In this section, we have discussed how to create a table and how to add new rows in the database.		
Fetching the data from record is simple as the inserting them. The execute method uses the SQL command of getting all the data from the table using "Select * from table_name" and all the table data can be fetched in an object in the form of list of lists.		
# Python code to demonstrate SQL to fetch data.		
# importing the module		
import sqlite3		
# connect with the myTable database		
connection = sqlite3.connect("myTable.db")		

# cursor object		
crsr = connection.cursor()		
# execute the command to fetch all the data from the table emp		
crsr.execute("SELECT * FROM emp")		
# store all the fetched data in the ans variable		
ans = crsr.fetchall()		
# Since we have already selected all the data entries		
# using the "SELECT *" SQL command and stored them in		
# the ans variable, all we need to do now is to print		
# out the ans variable		
print(ans)		
It should be noted that the database file that will be created will be in the same folder as that of the python file. If we wish to change the path of the file, change the path while opening the file.		
SQL using Python and SQLite Set 2		
SQL using Python Set 3 (Handling large data)		
This article is contributed by Rishabh Bansal . If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org . See your article appearing on the GeeksforGeeks main page and help other Geeks.		
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.		
Attention geek! Strengthen your foundations with the Python Programming Foundation Course and learn the basics.		
To begin with, your interview preparations Enhance your Data Structures concepts with the Python DS Course.		

<p>n this article, I have discussed how to connect to MySQL database remotely using python. For any application, it is very important to store the database on a server for easy data access. It is quite complicated to connect to the database remotely because every service provider doesn't provide remote access to the MySQL database. Here I am using python's MySQLdb module for connecting to our database which is at any server that provides remote access.</p>		
What is MySQLdb?		
<p>MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.</p>		
Packages to Install		
mysql-connector-python		
mysql-python		
How to connect to a remote MySQL database using python?		
<p>Before we start you should know the basics of SQL. Now let us discuss the methods used in this code:</p>		
<p>connect(): This method is used for creating a connection to our database it has four arguments:</p>		
1. Server Name		
2. Database User Name		
3. Database Provider		
4. Database Name		
<p>cursor(): This method creates a cursor object that is capable of executing SQL queries on the database.</p>		
<p>execute(): This method is used for executing SQL queries on the database. It takes a sql query(as string) as an argument.</p>		
<p>fetchone(): This method retrieves the next row of a query result set and returns a single sequence, or None if no more rows are available.</p>		
<p>close() : This method close the database connection.</p>		
Free remote mysql database providers:		

1.www.freemysqlhosting.net		
2.www.heliohost.org		
Python3		
'''This code would not be run on geeksforgeeks IDE because required module are not installed on IDE. Also this code requires a remote MySQL databaseconnection with valid Hostname, Dbusername Password and Dbname'''		
# Module For Connecting To MySQL database		
import MySQLdb		
# Function for connecting to MySQL database		
def mysqlconnect():		
#Trying to connect		
try:		
db_connection= MySQLdb.connect		
("Hostname","dbusername","password","dbname")		
# If connection is not successful		
except:		
print("Can't connect to database")		
return 0		
# If Connection Is Successful		
print("Connected")		
# Making Cursor Object For Query Execution		
cursor=db_connection.cursor()		
# Executing Query		
cursor.execute("SELECT CURDATE();")		
# Above Query Gives Us The Current Date		
# Fetching Data		
m = cursor.fetchone()		
# Printing Result Of Above		
print("Today's Date Is ",m[0])		

# Closing Database Connection		
db_connection.close()		
# Function Call For Connecting To Our Database		
mysqlconnect()		
Connected		
Today's Date Is 2017-11-14		
Python3		
# Python code to illustrate and create a		
# table in database		
import mysql.connector as mysql		
# Open database connection		
db = mysql.connect(host="localhost",user="root",password="tiger", database="python")		
cursor = db.cursor()		
# Drop table if it already exist using execute()		
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")		
# Create table as per requirement		
sql = "CREATE TABLE EMPLOYEE (FNAME CHAR(20) NOT NULL, LNAME CHAR(20), AGE INT)"		
cursor.execute(sql) #table created		
# disconnect from server		



```

MySQL 8.0 Command Line Client
mysql> desc employee;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| FNAME | char(20) | NO   |     | NULL    |       |
| LNAME | char(20) | YES  |     | NULL    |       |
| AGE   | int     | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+

```

```
3 rows in set (0.01 sec)
```


[illegible]

[illegible]

[illegible]

[illegible]

7
8
9
10

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as –

GadFly

mSQL

MySQL

PostgreSQL

Microsoft SQL Server 2000

Informix

Interbase

Oracle

Sybase

[Here is the list of available Python database interfaces: Python Database Interfaces and APIs. You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.](#)

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following –

Importing the API module.

Acquiring a connection with the database.

Issuing SQL statements and stored procedures.

Closing the connection

We would learn all the concepts using MySQL, so let us talk about MySQLdb module.

What is MySQLdb?

MySQLdb is an interface for connecting to a MySQL database server from Python. It implements the Python Database API v2.0 and is built on top of the MySQL C API.

How do I Install MySQLdb?

Before proceeding, you make sure you have MySQLdb installed on your machine. Just type the following in your Python script and execute it –

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

```
import MySQLdb
```

If it produces the following result, then it means MySQLdb module is not installed –

```
Traceback (most recent call last):
```

```
File "test.py", line 3, in <module>
```

```
import MySQLdb
```

```
ImportError: No module named MySQLdb
```

To install MySQLdb module, use the following command –

For Ubuntu, use the following command –

```
$ sudo apt-get install python-pip python-dev libmysqlclient-dev
```

For Fedora, use the following command –

```
$ sudo dnf install python python-devel mysql-devel redhat-rpm-config gcc
```

For Python command prompt, use the following command –

```
pip install MySQL-python
```

Note – Make sure you have root privilege to install above module.

Database Connection

Before connecting to a MySQL database, make sure of the followings –

You have created a database TESTDB.

You have created a table EMPLOYEE in TESTDB.

This table has fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.

User ID "testuser" and password "test123" are set to access TESTDB.

Python module MySQLdb is installed properly on your machine.

[You have gone through MySQL tutorial to understand MySQL Basics.](#)

Example

Following is the example of connecting with MySQL database "TESTDB"

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

```
import MySQLdb
```

```
# Open database connection
```

```
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )
```

```
# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute("SELECT VERSION() ")

# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print "Database version : %s " % data

# disconnect from server
db.close()
```

While running this script, it is producing the following result in my Linux machine.

```
Database version : 5.0.45
```

If a connection is established with the datasource, then a Connection Object is returned and saved into db for further use, otherwise db is set to None. Next, db object is used to create a cursor object, which in turn is used to execute SQL queries. Finally, before coming out, it ensures that database connection is closed and resources are released.

Creating Database Table

Once a database connection is established, we are ready to create tables or records into the database tables using execute method of the created cursor.

Example

Let us create Database table EMPLOYEE –

```
#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()
```

```

# Drop table if it already exist using execute() method.
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# Create table as per requirement
sql = """CREATE TABLE EMPLOYEE (
        FIRST_NAME  CHAR(20) NOT NULL,
        LAST_NAME   CHAR(20),
        AGE INT,
        SEX CHAR(1),
        INCOME FLOAT )"""

cursor.execute(sql)

# disconnect from server
db.close()

```

INSERT Operation

It is required when you want to create your records into a database table.

Example

The following example, executes SQL *INSERT* statement to create a record into *EMPLOYEE* table –

```

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
        LAST_NAME, AGE, SEX, INCOME)
        VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:
    # Execute the SQL command
    cursor.execute(sql)

```

```

# Commit your changes in the database
db.commit()
except:
# Rollback in case there is any error
db.rollback()

```

```

# disconnect from server
db.close()

```

Above example can be written as follows to create SQL queries dynamically –

```

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO EMPLOYEE(FIRST NAME, \
        LAST_NAME, AGE, SEX, INCOME) \
        VALUES ('%s', '%s', '%d', '%c', '%d' )" % \
        ('Mac', 'Mohan', 20, 'M', 2000)
try:
# Execute the SQL command
cursor.execute(sql)
# Commit your changes in the database
db.commit()
except:
# Rollback in case there is any error
db.rollback()

# disconnect from server
db.close()

```

Example

Following code segment is another form of execution where you can pass parameters directly –

```

.....
user_id = "test123"
password = "password"

con.execute('insert into Login values("%s", "%s")' % \
            (user_id, password))
.....

```

READ Operation

READ Operation on any database means to fetch some useful information from the database.

Once our database connection is established, you are ready to make a query into this database. You can use either fetchone() method to fetch single record or fetchall() method to fetch multiple values from a database table.

fetchone() – It fetches the next row of a query result set. A result set is an object that is returned when a cursor object is used to query a table.

fetchall() – It fetches all the rows in a result set. If some rows have already been extracted from the result set, then it retrieves the remaining rows from the result set.

rowcount – This is a read-only attribute and returns the number of rows that were affected by an execute() method.

Example

The following procedure queries all the records from EMPLOYEE table having salary more than 1000 –

```

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

sql = "SELECT * FROM EMPLOYEE \
       WHERE INCOME > '%d'" % (1000)
try:
    # Execute the SQL command
    cursor.execute(sql)

```



```

# Fetch all the rows in a list of lists.
results = cursor.fetchall()
for row in results:
    fname = row[0]
    lname = row[1]
    age = row[2]
    sex = row[3]
    income = row[4]
    # Now print fetched result
    print "fname=%s,lname=%s,age=%d,sex=%s,income=%d" % \
        (fname, lname, age, sex, income )
except:
    print "Error: unable to fetch data"

# disconnect from server
db.close()

```

This will produce the following result –

```
fname=Mac, lname=Mohan, age=20, sex=M, income=2000
```

Update Operation

UPDATE Operation on any database means to update one or more records, which are already available in the database.

The following procedure updates all the records having SEX as 'M'. Here, we increase AGE of all the males by one year.

Example

```

#!/usr/bin/python

import MySQLdb

# Open database connection

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to UPDATE required records

```

```

sql = "UPDATE EMPLOYEE SET AGE = AGE + 1
      WHERE SEX = '%c'" % ('M')
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error
    db.rollback()

# disconnect from server
db.close()

```

DELETE Operation

DELETE operation is required when you want to delete some records from your database. Following is the procedure to delete all the records from EMPLOYEE where AGE is more than 20 –

Example

```

#!/usr/bin/python

import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
try:
    # Execute the SQL command
    cursor.execute(sql)
    # Commit your changes in the database
    db.commit()
except:
    # Rollback in case there is any error

```

```
db.rollback()
```

```
# disconnect from server
```

```
db.close()
```

Performing Transactions

Transactions are a mechanism that ensures data consistency. Transactions have the following four properties –

Atomicity – Either a transaction completes or nothing happens at all.

Consistency – A transaction must start in a consistent state and leave the system in a consistent state.

Isolation – Intermediate results of a transaction are not visible outside the current transaction.

Durability – Once a transaction was committed, the effects are persistent, even after a system failure.

The Python DB API 2.0 provides two methods to either *commit* or *rollback* a transaction.

Example

You already know how to implement transactions. Here is again similar example –

```
# Prepare SQL query to DELETE required records
```

```
sql = "DELETE FROM EMPLOYEE WHERE AGE > '%d'" % (20)
```

```
try:
```

```
    # Execute the SQL command
```

```
    cursor.execute(sql)
```

```
    # Commit your changes in the database
```

```
    db.commit()
```

```
except:
```

```
    # Rollback in case there is any error
```

```
    db.rollback()
```

COMMIT Operation

Commit is the operation, which gives a green signal to database to finalize the changes, and after this operation, no change can be reverted back.

Here is a simple example to call commit method.

```
db.commit()
```

ROLLBACK Operation

If you are not satisfied with one or more of the changes and you want to revert back those changes completely, then use rollback() method.

Here is a simple example to call rollback() method.

```
db.rollback()
```

Disconnecting Database

To disconnect Database connection, use close() method.

```
db.close()
```

If the connection to a database is closed by the user with the close() method, any outstanding transactions are rolled back by the DB. However, instead of depending on any of DB lower level implementation details, your application would be better off calling commit or rollback explicitly.

Handling Errors

There are many sources of errors. A few examples are a syntax error in an executed SQL statement, a connection failure, or calling the fetch method for an already canceled or finished statement handle.

The DB API defines a number of errors that must exist in each database module. The following table lists these exceptions.

Exception & Description
Warning
Used for non-fatal issues. Must subclass StandardError.
Error
Base class for errors. Must subclass StandardError.
InterfaceError
Used for errors in the database module, not the database itself. Must subclass Error.
DatabaseError
Used for errors in the database. Must subclass Error.
DataError
Subclass of DatabaseError that refers to errors in the data.
OperationalError

Subclass of DatabaseError that refers to errors such as the loss of a connection to the database. These errors are generally outside of the control of the Python scripter.

IntegrityError

Subclass of DatabaseError for situations that would damage the relational integrity, such as uniqueness constraints or foreign keys.

InternalError

Subclass of DatabaseError that refers to errors internal to the database module, such as a cursor no longer being active.

ProgrammingError

Subclass of DatabaseError that refers to errors such as a bad table name and other things that can safely be blamed on you.

NotSupportedError

Subclass of DatabaseError that refers to trying to call unsupported functionality.

Your Python scripts should handle these errors, but before using any of the above exceptions, make sure your MySQLdb has support for that exception. You can get more information about them by reading the DB API 2.0 specification.

[illegible]

[illegible]

Declare a global variable inside a function, and use it outside the function:		
#create a function:		
def myfunction():		
global x		
x = "hello"		
#execute the function:		
myfunction()		
#x should now be global, and accessible in the global scope.		
print(x)		

	The sqlite3.Cursor class is an instance using which you can invoke methods that execute SQLite statements, fetch data from the result sets of the queries. You can create Cursor object using the cursor() method of the Connection object/class.
	Example
	<pre>import sqlite3</pre>
	<pre>#Connecting to sqlite</pre>
	<pre>conn = sqlite3.connect('example.db')</pre>
	<pre>#Creating a cursor object using the cursor() method</pre>
	<pre>cursor = conn.cursor()</pre>
	Methods
	Following are the various methods provided by the Cursor class/object.
Sr.No	Method & Description
1	execute()
	This routine executes an SQL statement. The SQL statement may be parameterized (i.e., placeholders instead of SQL literals). The pycopg2 module supports placeholder using %s sign
	For example:cursor.execute("insert into people values (%s, %s)", (who, age))
2	executemany()
	This routine executes an SQL command against all parameter sequences or mappings found in the sequence sql.
3	fetchone()
	This method fetches the next row of a query result set, returning a single sequence, or None when no more data is available.
4	fetchmany()
	This routine fetches the next set of rows of a query result, returning a list. An empty list is returned when no more rows are available. The method tries to fetch as many rows as indicated by the size parameter.
5	fetchall()

	This routine fetches all (remaining) rows of a query result, returning a list. An empty list is returned when no rows are available.
	Properties
	Following are the properties of the Cursor class –
Sr.No	Method & Description
1	arraySize
	This is a read/write property you can set the number of rows returned by the fetchmany() method.
2	description
	This is a read only property which returns the list containing the description of columns in a result-set.
3	lastrowid
	This is a read only property, if there are any auto-incremented columns in the table, this returns the value generated for that column in the last INSERT or, UPDATE operation.
4	rowcount
	This returns the number of rows returned/updated in case of SELECT and UPDATE operations.
5	connection
	This read-only attribute provides the SQLite database Connection used by the Cursor object.

The destroy() method in Tkinter destroys a widget. It is useful in controlling the behavior of various widgets which depend on each other. Also when a process is complete by some user action we need to destroy the GUI components to free the memory as well as clear the screen. The destroy() method achieves all this.

In the below example we have screen with 3 buttons. Clicking the first button closes the window itself where as the clicking of the second button closes the 1st button and so on. This behavior is emulated by using the destroy method as shown in the program below.

Example

```
from tkinter import *
from tkinter.ttk import *
#tkinter window
base = Tk()

#This button can close the window
button_1 = Button(base, text ="I close the Window", command =
base.destroy)
#External paddign for the buttons
button_1.pack(pady = 40)

#This button closes the first button
button_2 = Button(base, text ="I close the first button",
command = button_1.destroy)
button_2.pack(pady = 40)

#This button closes the second button
button_3 = Button(base, text ="I close the second button",
command = button_2.destroy)
button_3.pack(pady = 40)
mainloop()
```

Running the above code gives us the following result –

On clicking the various buttons we can observer different behaviors as mentioned in the program.

 tk



I close the first button				

resizable() method is used to allow Tkinter root window to change it's size according to the users need as well we can prohibit resizing of the Tkinter window.

So, basically, if user wants to create a fixed size window, this method can be used.

How to use:

-> import tkinter

-> root = Tk()

-> root.resizable(height = None, width = None)

Arguments to be passed:

-> In resizable() method user can pass either positive integer or True, to make the window resizable.

-> To make window non-resizable user can pass 0 or False.

Code #1: Allowing root window to change it's size

```
# importing only those functions
# which are needed
```

```
from tkinter import *
```

```
from tkinter.ttk import *
```

```
from time import strftime
```

```
# creating tkinter window
```

```
root = Tk()
```

```
root.title('Resizable')
```

```
root.geometry('250x100')
```

```
Label(root, text = 'It\'s resizable').pack(side = TOP,
pady = 10)
```

```
# Allowing root window to change
```

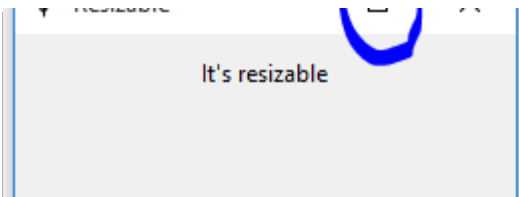
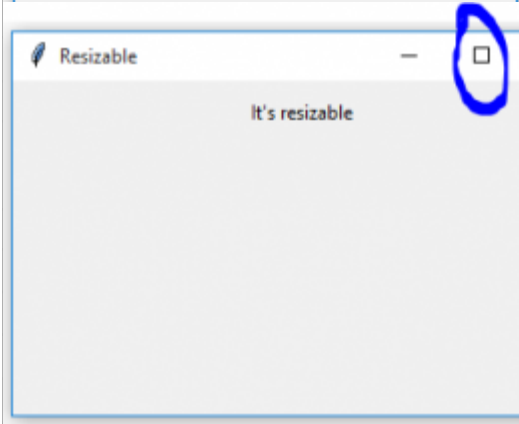
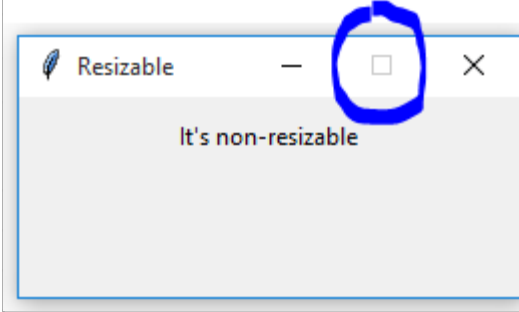
```
# it's size according to user's need
```

```
root.resizable(True, True)
```

```
mainloop()
```

Output:



	<p>at the part inside blue circle and can be expanded. ne blue is enabled so you</p>	
	<p>change it's size (fixed</p>	
<pre>from time import strftime</pre>		
<pre># creating tkinter window</pre>		
<pre>root = Tk()</pre>		
<pre>root.title('Resizable')</pre>		
<pre>root.geometry('250x100')</pre>		
<pre>Label(root, text = 'It\'s non-resizable').pack(side =</pre>		
<pre>TOP, pady = 10)</pre>		
<pre># Restricting root window to change</pre>		
<pre># it's size according to user's need</pre>		
<pre>root.resizable(0, 0)</pre>		
<pre>mainloop()</pre>		
	<p>de the blue circle is not be altered.</p>	

Python | pack() method in Tkinter

[Difficulty Level : Easy](#)

Last Updated : 12 Jan, 2021

The Pack geometry manager packs widgets in rows or columns. We can use options like **fill**, **expand**, and **side** to control this geometry manager.

Compared to the **grid** manager, the **pack** manager is somewhat limited, but it's much easier to use in a few, but quite common situations:

Put a widget inside a frame (or any other container widget), and have it fill the entire frame

Place a number of widgets on top of each other

Place a number of widgets side by side

Code #1: Putting a widget inside frame and filling entire frame. We can do this with the help of **expand** and **fill** options.

Python3

```
# Importing tkinter module
```

```
from tkinter import * from tkinter.ttk import *
```

```
# creating Tk window
```

```
master = Tk()
```

```
# creating a Frame which can expand according  
# to the size of the window
```

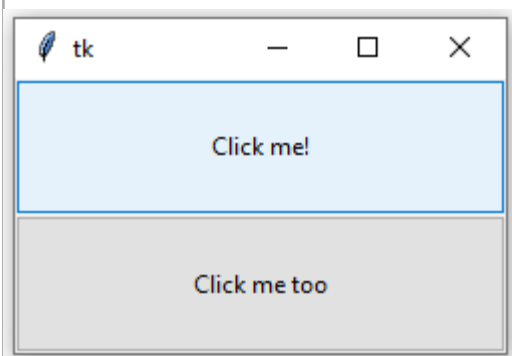
```
pane = Frame(master)
```

```
pane.pack(fill = BOTH, expand = True)
```

```
# button widgets which can also expand and fill  
# in the parent widget entirely
```

# Button 1			
b1 = Button(pane, text = "Click me !")			
b1.pack(fill = BOTH, expand = True)			
# Button 2			
b2 = Button(pane, text = "Click me too")			
b2.pack(fill = BOTH, expand = True)			
# Execute Tkinter			
master.mainloop()			

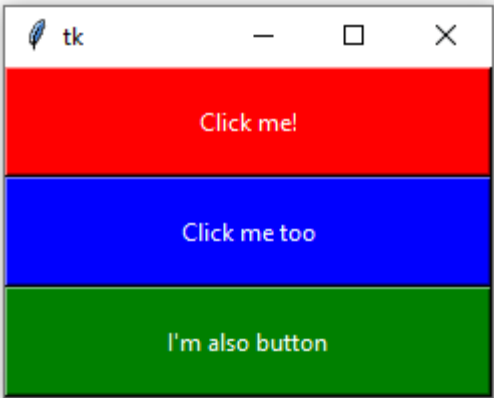
Output:




of each other and
side option.

Python3

# Importing tkinter module			
from tkinter import *			
# from tkinter.ttk import *			
# creating Tk window			
master = Tk()			
# cretaing a Fra, e which can expand according			
# to the size of the window			
pane = Frame(master)			
pane.pack(fill = BOTH, expand = True)			
# button widgets which can also expand and fill			
# in the parent widget entirely			
# Button 1			

b1 = Button(pane, text = "Click me !",			
background = "red", fg = "white")			
b1.pack(side = TOP, expand = True, fill = BOTH)			
# Button 2			
b2 = Button(pane, text = "Click me too",			
background = "blue", fg = "white")			
b2.pack(side = TOP, expand = True, fill = BOTH)			
# Button 3			
b3 = Button(pane, text = "I'm also button",			
background = "green", fg = "white")			
b3.pack(side = TOP, expand = True, fill = BOTH)			
# Execute Tkinter			
master.mainloop()			
Output:			
			
# from tkinter.ttk import *			
# creating Tk window			
master = Tk()			
# creating a Frame which can expand according			
# to the size of the window			
pane = Frame(master)			

pane.pack(fill = BOTH, expand = True)			
# button widgets which can also expand and fill			
# in the parent widget entirely			
# Button 1			
b1 = Button(pane, text = "Click me !",			
background = "red", fg = "white")			
b1.pack(side = LEFT, expand = True, fill = BOTH)			
# Button 2			
b2 = Button(pane, text = "Click me too",			
background = "blue", fg = "white")			
b2.pack(side = LEFT, expand = True, fill = BOTH)			
# Button 3			
b3 = Button(pane, text = "I'm also button",			
background = "green", fg = "white")			
b3.pack(side = LEFT, expand = True, fill = BOTH)			
# Execute Tkinter			
master.mainloop()			
			

[illegible]

[illegible]

[Defining a Set](#)

Python's built-in set type has the following characteristics:

Sets are unordered.

Set elements are unique. Duplicate elements are not allowed.

A set itself may be modified, but the elements contained in the set must be of an immutable type.

Let's see what all that means, and how you can work with sets in Python.

A set can be created in two ways. First, you can define a set with the built-in `set()` function:

```
x = set(<iter>)
```

In this case, the argument `<iter>` is an iterable—again, for the moment, think list or tuple—that generates the list of objects to be included in the set. This is analogous to the `<iter>` argument given to the `.extend()` list method:

```
>>>
```

```
>>> x = set(['foo', 'bar', 'baz', 'foo', 'qux'])
```

```
>>> x
```

```
{'qux', 'foo', 'bar', 'baz'}
```

```
>>> x = set(('foo', 'bar', 'baz', 'foo', 'qux'))
```

```
>>> x
```

```
{'qux', 'foo', 'bar', 'baz'}
```

[Strings are also iterable, so a string can be passed to `set\(\)` as well. You have already seen that `list\(s\)` generates a list of the characters in the string `s`. Similarly, `set\(s\)` generates a set of the characters in `s`:](#)

```
>>>
```

```
>>> s = 'quux'
```

```
>>> list(s)
```

```
['q', 'u', 'u', 'x']
```

```
>>> set(s)
```

```
{'x', 'u', 'q'}
```

You can see that the resulting sets are unordered: the original order, as specified in the definition, is not necessarily preserved. Additionally, duplicate values are only represented in the set once, as with the string `'foo'` in the first two examples and the letter `'u'` in the third.

Alternately, a set can be defined with curly braces (`{}`):

```
x = {<obj>, <obj>, ..., <obj>}
```

When a set is defined this way, each `<obj>` becomes a distinct element of the set, even if it is an iterable. This behavior is similar to that of the `.append()` list method.

Thus, the sets shown above can also be defined like this:

```
>>>
>>> x = {'foo', 'bar', 'baz', 'foo', 'qux'}
>>> x
{'qux', 'foo', 'bar', 'baz'}

>>> x = {'q', 'u', 'u', 'x'}
>>> x
{'x', 'q', 'u'}
```

To recap:

The argument to `set()` is an iterable. It generates a list of elements to be placed into the set.

The objects in curly braces are placed into the set intact, even if they are iterable.

Observe the difference between these two set definitions:

```
>>>
>>> {'foo'}
{'foo'}

>>> set('foo')
{'o', 'f'}
```

A set can be empty. However, recall that Python interprets empty curly braces (`{}`) as an empty dictionary, so the only way to define an empty set is with the `set()` function:

```
>>>
>>> x = set()
>>> type(x)
<class 'set'>

>>> x
set()
```



```

>>> x = {}
>>> type(x)
<class 'dict'>
An empty set is falsy in a Boolean context:

>>>
>>> x = set()
>>> bool(x)
FALSE
>>> x or 1
1
>>> x and 1
set()
You might think the most intuitive sets would contain similar objects—for example,
even numbers or surnames:

>>>
>>> s1 = {2, 4, 6, 8, 10}
>>> s2 = {'Smith', 'McArthur', 'Wilson', 'Johansson'}

Python does not require this, though. The elements in a set can be objects of different types:

>>>
>>> x = {42, 'foo', 3.14159, None}
>>> x
{None, 'foo', 42, 3.14159}

Don't forget that set elements must be immutable. For example, a tuple may be included in a
set:

>>>
>>> x = {42, 'foo', (1, 2, 3), 3.14159}
>>> x
{42, 'foo', 3.14159, (1, 2, 3)}

But lists and dictionaries are mutable, so they can't be set elements:

>>>
>>> a = [1, 2, 3]
>>> {a}
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    {a}
TypeError: unhashable type: 'list'

>>> d = {'a': 1, 'b': 2}
>>> {d}

```

Traceback (most recent call last):	
File "<pyshell#72>", line 1, in <module>	
{d}	
TypeError: unhashable type: 'dict'	
Remove ads	
Set Size and Membership	
The len() function returns the number of elements in a set, and the in and not in operators can be used to test for membership:	
>>>	
>>> x = {'foo', 'bar', 'baz'}	
>>> len(x)	
3	
>>> 'bar' in x	
TRUE	
>>> 'qux' in x	
FALSE	
Operating on a Set	
Many of the operations that can be used for Python's other composite data types don't make sense for sets. For example, sets can't be indexed or sliced. However, Python provides a whole host of operations on set objects that generally mimic the operations that are defined for mathematical sets.	
Operators vs. Methods	
Most, though not quite all, set operations in Python can be performed in two different ways: by operator or by method. Let's take a look at how these operators and methods work, using set union as an example.	
Given two sets, x1 and x2, the union of x1 and x2 is a set consisting of all elements in either set.	
Consider these two sets:	
x1 = {'foo', 'bar', 'baz'}	
x2 = {'baz', 'qux', 'quux'}	
The union of x1 and x2 is {'foo', 'bar', 'baz', 'qux', 'quux'}.	

Note: Notice that the element 'baz', which appears in both x1 and x2, appears only once in the union. Sets never contain duplicate values.

In Python, set union can be performed with the | operator:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'baz', 'qux', 'quux'}
```

```
>>> x1 | x2
```

```
{'baz', 'quux', 'qux', 'bar', 'foo'}
```

Set union can also be obtained with the .union() method. The method is invoked on one of the sets, and the other is passed as an argument:

```
>>>
```

```
>>> x1.union(x2)
```

```
{'baz', 'quux', 'qux', 'bar', 'foo'}
```

The way they are used in the examples above, the operator and method behave identically. But there is a subtle difference between them. When you use the | operator, both operands must be sets. The .union() method, on the other hand, will take any iterable as an argument, convert it to a set, and then perform the union.

Observe the difference between these two statements:

```
>>>
```

```
>>> x1 | ('baz', 'qux', 'quux')
```

```
Traceback (most recent call last):
```

```
File "<pysHELL#43>", line 1, in <module>
```

```
x1 | ('baz', 'qux', 'quux')
```

```
TypeError: unsupported operand type(s) for |: 'set' and 'tuple'
```

```
>>> x1.union(('baz', 'qux', 'quux'))
```

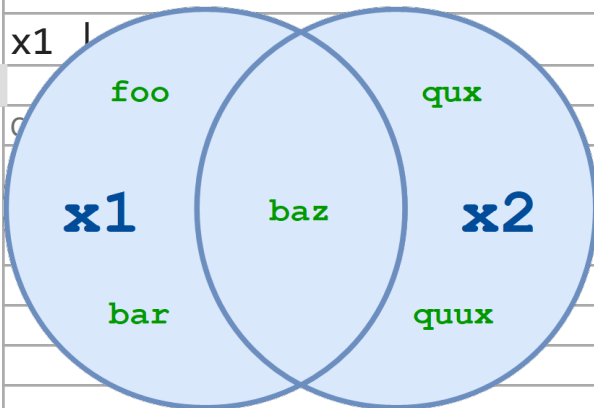
```
{'baz', 'quux', 'qux', 'bar', 'foo'}
```

Both attempt to compute the union of x1 and the tuple ('baz', 'qux', 'quux'). This fails with the | operator but succeeds with the .union() method.

[Available Operators and Methods](#)

Below is a list of the set operations available in Python. Some are performed by operator, some by method, and some by both. The principle outlined above generally applies: where a set is expected, methods will typically accept any iterable as an argument, but operators require actual sets as operands.

```
x1.union(x2[, x3 ...])
```



`x1.union(x2)` and `x1 | x2` both return the set of all elements in either `x1` or `x2`:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'baz', 'quux', 'quux'}
```

```
>>> x1.union(x2)
```

```
{'foo', 'quux', 'quux', 'baz', 'bar'}
```

```
>>> x1 | x2
```

```
{'foo', 'quux', 'quux', 'baz', 'bar'}
```

More than two sets may be specified with either the operator or the method:

```
>>>
```

```
>>> a = {1, 2, 3, 4}
```

```
>>> b = {2, 3, 4, 5}
```

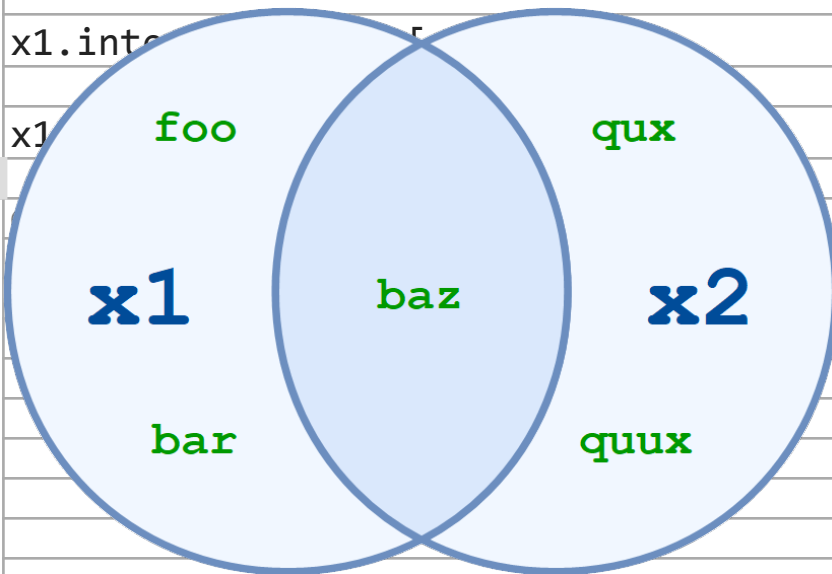
```
>>> c = {3, 4, 5, 6}
```

```
>>> d = {4, 5, 6, 7}
```

```
>>> a.union(b, c, d)
{1, 2, 3, 4, 5, 6, 7}
```

```
>>> a | b | c | d
{1, 2, 3, 4, 5, 6, 7}
```

The resulting set contains all elements that are present in any of the specified sets.



`x1.intersection(x2)` and `x1 & x2` return the set of elements common to both `x1` and `x2`:

```
>>>
>>> x1 = {'foo', 'bar', 'baz'}
>>> x2 = {'baz', 'qux', 'quux'}
```

```
>>> x1.intersection(x2)
{'baz'}
```

```
>>> x1 & x2
{'baz'}
```

You can specify multiple sets with the intersection method and operator, just like you can with set union:

```
>>>
```

```
>>> a = {1, 2, 3, 4}
```

```
>>> b = {2, 3, 4, 5}
```

```
>>> c = {3, 4, 5, 6}
```

```
>>> d = {4, 5, 6, 7}
```

```
>>> a.intersection(b, c, d)
```

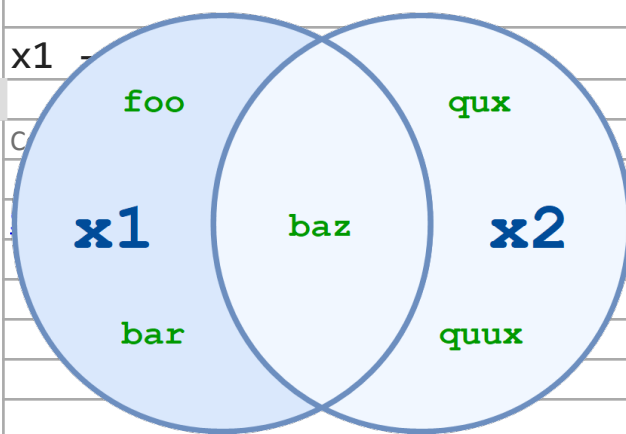
```
{4}
```

```
>>> a & b & c & d
```

```
{4}
```

The resulting set contains only elements that are present in all of the specified sets.

```
x1.difference(x2[, x3 ...])
```



`x1.difference(x2)` and `x1 - x2` return the set of all elements that are in `x1` but not in `x2`:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'baz', 'qux', 'quux'}
```

```
>>> x1.difference(x2)
```

```
{'foo', 'bar'}
```

```
>>> x1 - x2
{'foo', 'bar'}
```

Another way to think of this is that `x1.difference(x2)` and `x1 - x2` return the set that results when any elements in `x2` are removed or subtracted from `x1`.

Once again, you can specify more than two sets:

```
>>>
>>> a = {1, 2, 3, 30, 300}
>>> b = {10, 20, 30, 40}
>>> c = {100, 200, 300, 400}

>>> a.difference(b, c)
{1, 2, 3}

>>> a - b - c
{1, 2, 3}
```

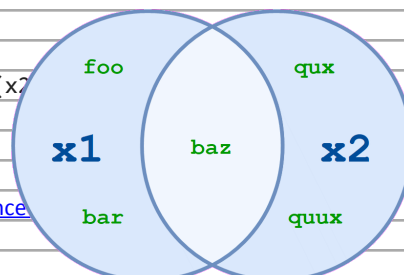
When multiple sets are specified, the operation is performed from left to right. In the example above, `a - b` is computed first, resulting in `{1, 2, 3, 300}`. Then `c` is subtracted from that set, leaving `{1, 2, 3}`.

```
x1.symmetric_difference(x2)
```

```
x1 ^ x2 [^ x3 ...]
```

[Compute the symmetric difference.](#)

[Set Symmetric Difference](#)



`x1.symmetric_difference(x2)` and `x1 ^ x2` return the set of all elements in either `x1` or `x2`, but not both:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'baz', 'qux', 'quux'}
```

```
>>> x1.symmetric_difference(x2)
```

```
{'foo', 'qux', 'quux', 'bar'}
```

```
>>> x1 ^ x2
```

```
{'foo', 'qux', 'quux', 'bar'}
```

The `^` operator also allows more than two sets:

```
>>>
```

```
>>> a = {1, 2, 3, 4, 5}
```

```
>>> b = {10, 2, 3, 4, 50}
```

```
>>> c = {1, 50, 100}
```

```
>>> a ^ b ^ c
```

```
{100, 5, 10}
```

As with the difference operator, when multiple sets are specified, the operation is performed from left to right.

Curiously, although the `^` operator allows multiple sets, the `.symmetric_difference()` method doesn't:

```
>>>
```

```
>>> a = {1, 2, 3, 4, 5}
```

```
>>> b = {10, 2, 3, 4, 50}
```

```
>>> c = {1, 50, 100}
```

```
>>> a.symmetric_difference(b, c)
```

Traceback (most recent call last):

File "<pyshell#11>", line 1, in <module>

a.symmetric_difference(b, c)

TypeError: symmetric_difference() takes exactly one argument (2 given)

`x1.isdisjoint(x2)`

Determines whether or not two sets have any elements in common.

`x1.isdisjoint(x2)` returns True if `x1` and `x2` have no elements in common:


```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'baz', 'qux', 'quux'}
```

```
>>> x1.isdisjoint(x2)
```

```
FALSE
```

```
>>> x2 - {'baz'}
```

```
{'quux', 'qux'}
```

```
>>> x1.isdisjoint(x2 - {'baz'})
```

```
TRUE
```

If `x1.isdisjoint(x2)` is True, then `x1 & x2` is the empty set:

```
>>>
```

```
>>> x1 = {1, 3, 5}
```

```
>>> x2 = {2, 4, 6}
```

```
>>> x1.isdisjoint(x2)
```

```
TRUE
```

```
>>> x1 & x2
```

```
set()
```

Note: There is no operator that corresponds to the `.isdisjoint()` method.

`x1.issubset(x2)`

`x1 <= x2`

Determine whether one set is a subset of the other.

In set theory, a set `x1` is considered a subset of another set `x2` if every element of `x1` is in `x2`.

`x1.issubset(x2)` and `x1 <= x2` return True if `x1` is a subset of `x2`:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x1.issubset({'foo', 'bar', 'baz', 'qux', 'quux'})
```

```
TRUE
```

```
>>> x2 = {'baz', 'qux', 'quux'}
```

```
>>> x1 <= x2
```

```
FALSE
```

A set is considered to be a subset of itself:

```
>>>
```

```
>>> x = {1, 2, 3, 4, 5}
```

```
>>> x.issubset(x)
```

```
TRUE
```

```
>>> x <= x
```

```
TRUE
```

It seems strange, perhaps. But it fits the definition—every element of x is in x.

x1 < x2

Determines whether one set is a proper subset of the other.

A proper subset is the same as a subset, except that the sets can't be identical. A set x1 is considered a proper subset of another set x2 if every element of x1 is in x2, and x1 and x2 are not equal.

x1 < x2 returns True if x1 is a proper subset of x2:

```
>>>
```

```
>>> x1 = {'foo', 'bar'}
```

```
>>> x2 = {'foo', 'bar', 'baz'}
```

```
>>> x1 < x2
```

```
TRUE
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'foo', 'bar', 'baz'}
```

```
>>> x1 < x2
```

```
FALSE
```

While a set is considered a subset of itself, it is not a proper subset of itself:

```
>>>
```

```
>>> x = {1, 2, 3, 4, 5}
```

```
>>> x <= x
```

```
TRUE
```

```
>>> x < x
```

```
FALSE
```

Note: The < operator is the only way to test whether a set is a proper subset. There is no corresponding method.

x1.issuperset(x2)

x1 >= x2

Determine whether one set is a superset of the other.

A superset is the reverse of a subset. A set x1 is considered a superset of another set x2 if x1 contains every element of x2.

x1.issuperset(x2) and x1 >= x2 return True if x1 is a superset of x2:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x1.issuperset({'foo', 'bar'})
```

```
TRUE
```

```
>>> x2 = {'baz', 'qux', 'quux'}
```

```
>>> x1 >= x2
```

```
FALSE
```

You have already seen that a set is considered a subset of itself. A set is also considered a superset of itself:

```
>>>
```

```
>>> x = {1, 2, 3, 4, 5}
```

```
>>> x.issuperset(x)
```

```
TRUE
```

```
>>> x >= x
```

```
TRUE
```

x1 > x2

Determines whether one set is a proper superset of the other.

A proper superset is the same as a superset, except that the sets can't be identical. A set x1 is considered a proper superset of another set x2 if x1 contains every element of x2, and x1 and x2 are not equal.

x1 > x2 returns True if x1 is a proper superset of x2:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'foo', 'bar'}
```

```
>>> x1 > x2
```

```
TRUE
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'foo', 'bar', 'baz'}
```

```
>>> x1 > x2
```

```
FALSE
```

A set is not a proper superset of itself:	
>>>	
>>> x = {1, 2, 3, 4, 5}	
>>> x > x	
FALSE	
Note: The > operator is the only way to test whether a set is a proper superset. There is no corresponding method.	
Remove ads	
Modifying a Set	
Although the elements contained in a set must be of immutable type, sets themselves can be modified. Like the operations above, there are a mix of operators and methods that can be used to change the contents of a set.	
Augmented Assignment Operators and Methods	
Each of the union, intersection, difference, and symmetric difference operators listed above has an augmented assignment form that can be used to modify a set. For each, there is a corresponding method as well.	
x1.update(x2[, x3 ...])	
x1 = x2 [x3 ...]	
Modify a set by union.	
x1.update(x2) and x1 = x2 add to x1 any elements in x2 that x1 does not already have:	
>>>	
>>> x1 = {'foo', 'bar', 'baz'}	
>>> x2 = {'foo', 'baz', 'qux'}	
>>> x1 = x2	
>>> x1	
{'qux', 'foo', 'bar', 'baz'}	
>>> x1.update(['corge', 'garply'])	
>>> x1	
{'qux', 'corge', 'garply', 'foo', 'bar', 'baz'}	

<code>x1.intersection_update(x2[, x3 ...])</code>	
<code>x1 &= x2 [& x3 ...]</code>	
Modify a set by intersection.	
<code>x1.intersection_update(x2)</code> and <code>x1 &= x2</code> update <code>x1</code> , retaining only elements found in both <code>x1</code> and <code>x2</code> :	
<pre>>>> >>> x1 = {'foo', 'bar', 'baz'} >>> x2 = {'foo', 'baz', 'qux'} >>> x1 &= x2 >>> x1 {'foo', 'baz'}</pre>	
<pre>>>> x1.intersection_update(['baz', 'qux']) >>> x1 {'baz'}</pre>	
<code>x1.difference_update(x2[, x3 ...])</code>	
<code>x1 -= x2 [x3 ...]</code>	
Modify a set by difference.	
<code>x1.difference_update(x2)</code> and <code>x1 -= x2</code> update <code>x1</code> , removing elements found in <code>x2</code> :	
<pre>>>> >>> x1 = {'foo', 'bar', 'baz'} >>> x2 = {'foo', 'baz', 'qux'} >>> x1 -= x2 >>> x1 {'bar'}</pre>	
<pre>>>> x1.difference_update(['foo', 'bar', 'qux']) >>> x1 set()</pre>	
<code>x1.symmetric_difference_update(x2)</code>	

x1 ^= x2

Modify a set by symmetric difference.

x1.symmetric_difference_update(x2) and x1 ^= x2 update x1, retaining elements found in either x1 or x2, but not both:

```
>>>
```

```
>>> x1 = {'foo', 'bar', 'baz'}
```

```
>>> x2 = {'foo', 'baz', 'qux'}
```

```
>>>
```

```
>>> x1 ^= x2
```

```
>>> x1
```

```
{'bar', 'qux'}
```

```
>>>
```

```
>>> x1.symmetric_difference_update(['qux', 'corge'])
```

```
>>> x1
```

```
{'bar', 'corge'}
```

[Remove ads](#)

[Other Methods For Modifying Sets](#)

Aside from the augmented operators above, Python supports several additional methods that modify sets.

x.add(<elem>)

Adds an element to a set.

x.add(<elem>) adds <elem>, which must be a single immutable object, to x:

```
>>>
```

```
>>> x = {'foo', 'bar', 'baz'}
```

```
>>> x.add('qux')
```

```
>>> x
```

```
{'bar', 'baz', 'foo', 'qux'}
```

x.remove(<elem>)

Removes an element from a set.

`x.remove(<elem>)` removes `<elem>` from `x`. Python raises an exception if `<elem>` is not in `x`:

```
>>>
```

```
>>> x = {'foo', 'bar', 'baz'}
```

```
>>> x.remove('baz')
```

```
>>> x
```

```
{'bar', 'foo'}
```

```
>>> x.remove('qux')
```

```
Traceback (most recent call last):
```

```
File "<pysHELL#58>", line 1, in <module>
```

```
x.remove('qux')
```

```
KeyError: 'qux'
```

`x.discard(<elem>)`

Removes an element from a set.

`x.discard(<elem>)` also removes `<elem>` from `x`. However, if `<elem>` is not in `x`, this method quietly does nothing instead of raising an exception:

```
>>>
```

```
>>> x = {'foo', 'bar', 'baz'}
```

```
>>> x.discard('baz')
```

```
>>> x
```

```
{'bar', 'foo'}
```

```
>>> x.discard('qux')
```

```
>>> x
```

```
{'bar', 'foo'}
```

`x.pop()`

Removes a random element from a set.

`x.pop()` removes and returns an arbitrarily chosen element from `x`. If `x` is empty, `x.pop()` raises an exception:

```
>>>
```

```
>>> x = {'foo', 'bar', 'baz'}
```

```
>>> x.pop()
```

'bar'	
>>> x	
{'baz', 'foo'}	
>>> x.pop()	
'baz'	
>>> x	
{'foo'}	
>>> x.pop()	
'foo'	
>>> x	
set()	
>>> x.pop()	
Traceback (most recent call last):	
File "<pyshell#82>", line 1, in <module>	
x.pop()	
KeyError: 'pop from an empty set'	
x.clear()	
Clears a set.	
x.clear() removes all elements from x:	
>>>	
>>> x = {'foo', 'bar', 'baz'}	
>>> x	
{'foo', 'bar', 'baz'}	
>>>	
>>> x.clear()	
>>> x	
set()	
Remove ads	
Frozen Sets	
Python provides another built-in type called a frozenset , which is in all respects exactly like a set, except that a frozenset is immutable. You can perform non-modifying operations on a frozenset:	
>>>	
>>> x = frozenset(['foo', 'bar', 'baz'])	


```

>>> x
frozenset({'foo', 'baz', 'bar'})

>>> len(x)
3

>>> x & {'baz', 'qux', 'quux'}
frozenset({'baz'})
But methods that attempt to modify a frozenset fail:

>>>
>>> x = frozenset(['foo', 'bar', 'baz'])

>>> x.add('qux')
Traceback (most recent call last):
  File "<pyshell#127>", line 1, in <module>
    x.add('qux')
AttributeError: 'frozenset' object has no attribute 'add'

>>> x.pop()
Traceback (most recent call last):
  File "<pyshell#129>", line 1, in <module>
    x.pop()
AttributeError: 'frozenset' object has no attribute 'pop'

>>> x.clear()
Traceback (most recent call last):
  File "<pyshell#131>", line 1, in <module>
    x.clear()
AttributeError: 'frozenset' object has no attribute 'clear'

>>> x
frozenset({'foo', 'bar', 'baz'})

```

Deep Dive: Frozensets and Augmented Assignment

Since a frozenset is immutable, you might think it can't be the target of an augmented assignment operator. But observe:

```

>>>
>>> f = frozenset(['foo', 'bar', 'baz'])
>>> s = {'baz', 'qux', 'quux'}

>>> f &= s
>>> f

```

```
frozenset({'baz'})
```

What gives?

Python does not perform augmented assignments on frozensets in place. The statement `x &= s` is effectively equivalent to `x = x & s`. It isn't modifying the original `x`. It is reassigning `x` to a new object, and the object `x` originally referenced is gone.

You can verify this with the `id()` function:

```
>>>
```

```
>>> f = frozenset(['foo', 'bar', 'baz'])
```

```
>>> id(f)
```

```
56992872
```

```
>>> s = {'baz', 'qux', 'quux'}
```

```
>>> f &= s
```

```
>>> f
```

```
frozenset({'baz'})
```

```
>>> id(f)
```

```
56992152
```

`f` has a different integer identifier following the augmented assignment. It has been reassigned, not modified in place.

Some objects in Python are modified in place when they are the target of an augmented assignment operator. But frozensets aren't.

Frozensets are useful in situations where you want to use a set, but you need an immutable object. For example, you can't define a set whose elements are also sets, because set elements must be immutable:

```
>>>
```

```
>>> x1 = set(['foo'])
```

```
>>> x2 = set(['bar'])
```

```
>>> x3 = set(['baz'])
```

```
>>> x = {x1, x2, x3}
```

```
Traceback (most recent call last):
```

```
  File "<pyshe11#38>", line 1, in <module>
```

```
    x = {x1, x2, x3}
```

```
TypeError: unhashable type: 'set'
```

If you really feel compelled to define a set of sets (hey, it could happen), you can do it if the elements are frozensets, because they are immutable:

```
>>>
```

```
>>> x1 = frozenset(['foo'])
```

```
>>> x2 = frozenset(['bar'])
```

```
>>> x3 = frozenset(['baz'])
```

```
>>> x = {x1, x2, x3}
```

```
>>> x
```

```
{frozenset({'bar'}), frozenset({'baz'}), frozenset({'foo'})}
```

[Likewise, recall from the previous tutorial on dictionaries that a dictionary key must be immutable. You can't use the built-in set type as a dictionary key:](#)

```
>>>
```

```
>>> x = {1, 2, 3}
```

```
>>> y = {'a', 'b', 'c'}
```

```
>>>
```

```
>>> d = {x: 'foo', y: 'bar'}
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#3>", line 1, in <module>
```

```
    d = {x: 'foo', y: 'bar'}
```

```
TypeError: unhashable type: 'set'
```

If you find yourself needing to use sets as dictionary keys, you can use frozensets:

```
>>>
```

```
>>> x = frozenset({1, 2, 3})
```

```
>>> y = frozenset({'a', 'b', 'c'})
```

```
>>>
```

```
>>> d = {x: 'foo', y: 'bar'}
```

```
>>> d
```

```
{frozenset({1, 2, 3}): 'foo', frozenset({'c', 'a', 'b'}): 'bar'}
```

[Conclusion](#)

In this tutorial, you learned how to define **set** objects in Python, and you became familiar with the functions, operators, and methods that can be used to work with sets.

You should now be comfortable with the basic built-in data types that Python provides.

Next, you will begin to explore how the code that operates on those objects is organized and structured in a Python program.

Python-Tkinter Treeview scrollbar

Python has several options for constructing GUI and python tkinter is one of them. It is the standard GUI library for Python, which helps in making GUI applications easily. It provides an efficient object-oriented interface to the tk GUI toolkit. It also has multiple controls called widgets like text boxes, scrollbars, buttons, etc. Moreover, Tkinter has some geometry management methods namely, pack(), grid(), and, place() which are helpful in organizing widgets.

Note: For more information, refer to Python GUI – tkinter

Treeview scrollbar

When a scrollbar uses *treeview* widgets, then that type of scrollbar is called as *treeview scrollbar*. Where, a treeview widget is helpful in displaying more than one feature of every item listed in the tree to the right side of the tree in the form of columns. However, it can be implemented using tkinter in python with the help of some widgets and geometry management methods as supported by tkinter.

Below example illustrates the usage of **Treeview Scrollbar** using Python-tkinter:

Example 1:

```
# Python program to illustrate the usage of  
# treeview scrollbars using tkinter
```

```
from tkinter import ttk  
import tkinter as tk
```

```
# Creating tkinter window  
window = tk.Tk()  
window.resizable(width = 1, height = 1)
```

```
# Using treeview widget
```

```
treev = ttk.Treeview(window, selectmode ='browse')
```

```
# Calling pack method w.r.to treeview
```

```
treev.pack(side ='right')
```

```
# Constructing vertical scrollbar
```

```
# with treeview
```

```
versclbar = ttk.Scrollbar(window,
```

```
orient ="vertical",
```

```
command = treev.yview)
```

```
# Calling pack method w.r.to vertical
```

```
# scrollbar
```

```
versclbar.pack(side ='right', fill ='x')
```

```
# Configuring treeview
```

```
treev.configure(xscrollcommand = versclbar.set)
```

```
# Defining number of columns
```

```
treev["columns"] = ("1", "2", "3")
```

```
# Defining heading
```

```
treev['show'] = 'headings'
```

```
# Assigning the width and anchor to the
```

```
# respective columns
```

```
treev.column("1", width = 90, anchor ='c')
```

```
treev.column("2", width = 90, anchor ='se')
```

```
treev.column("3", width = 90, anchor ='se')
```

```
# Assigning the heading names to the
```

```
# respective columns
```

```
treev.heading("1", text ="Name")
```

```
treev.heading("2", text ="Sex")
```

```
treev.heading("3", text ="Age")
```

```
# Inserting the items and their features to the  
# columns built
```

```
treev.insert("", 'end', text = "L1",  
              values = ("Nidhi", "F", "25"))  
treev.insert("", 'end', text = "L2",  
              values = ("Nisha", "F", "23"))  
treev.insert("", 'end', text = "L3",  
              values = ("Preeti", "F", "27"))  
treev.insert("", 'end', text = "L4",  
              values = ("Rahul", "M", "20"))  
treev.insert("", 'end', text = "L5",  
              values = ("Sonu", "F", "18"))  
treev.insert("", 'end', text = "L6",  
              values = ("Rohit", "M", "19"))  
treev.insert("", 'end', text = "L7",  
              values = ("Geeta", "F", "25"))  
treev.insert("", 'end', text = "L8",  
              values = ("Ankit", "M", "22"))  
treev.insert("", 'end', text = "L10",  
              values = ("Mukul", "F", "25"))  
treev.insert("", 'end', text = "L11",  
              values = ("Mohit", "M", "16"))  
treev.insert("", 'end', text = "L12",  
              values = ("Vivek", "M", "22"))  
treev.insert("", 'end', text = "L13",  
              values = ("Suman", "F", "30"))
```

```
# Calling mainloop
```

```
window.mainloop()
```

Output:

Video Player

0:00

0:14

In the above program, we have used *pack()* method of the geometry management methods. And, we have constructed only vertical scrollbar as per the requirement of the code but you can construct both the bars as per your requirements. Moreover, anchors are used here in order to define the positions of the text. However, you can also use other geometry management methods in order to construct the treeview scrollbar.

--	--	--	--

Python Dictionary get() Method

[← Dictionary Methods](#)

Example

Get the value of the "model" item:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
x = car.get("model")
```

```
print(x)
```

[Try it Yourself »](#)

Definition and Usage

The `get()` method returns the value of the item with the specified key.

Syntax

```
dictionary.get(keyname, value)
```

Parameter Values

Parameter

keyname

value

More Examples

Example

Try to return the value of an item that do not exist:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
x = car.get("price", 15000)
```

```
print(x)
```

[illegible]

Python String find()

[Difficulty Level : Basic](#)

Last Updated : 09 May, 2020

The find() method returns the lowest index of the substring if it is found in given string. If its is not found then it returns -1.

Syntax :

str.find(sub,start,end)

Parameters :

sub : It's the substring which needs to be searched in the given string.

start : Starting position where sub is needs to be checked within the string.

end : Ending position where suffix is needs to be checked within the string.

NOTE : If start and end indexes are not provided then by default it takes 0 and length-1 as starting and ending indexes where ending indexes is not included in our search.

Returns:

returns the lowest index of the substring if it is found in given string. If it's not found then it returns -1.

[The find\(\) method is similar to index\(\). The only difference is find\(\) returns -1 if searched string is not found and index\(\) throws an exception in this case.](#)

CODE 1

```
word = 'geeks for geeks'
```

```
# returns first occurrence of Substring
```

```
result = word.find('geeks')
```

```
print ("Substring 'geeks' found at index:", result )
```

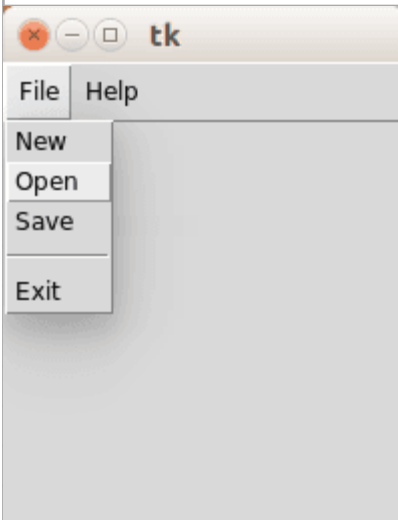
```
result = word.find('for')
```

```
print ("Substring 'for ' found at index:", result )
```

<pre># How to use find()</pre>
<pre>if (word.find('pawan') != -1):</pre>
<pre> print ("Contains given substring ")</pre>
<pre>else:</pre>
<pre> print ("Doesn't contains given substring")</pre>
Output:
Substring 'geeks' found at index: 0
Substring 'for ' found at index: 6
Doesn't contains given substring
CODE 2
<pre>word = 'geeks for geeks'</pre>
<pre># Substring is searched in 'eks for geeks'</pre>
<pre>print(word.find('ge', 2))</pre>
<pre># Substring is searched in 'eks for geeks'</pre>
<pre>print(word.find('geeks ', 2))</pre>
<pre># Substring is searched in 's for g'</pre>
<pre>print(word.find('g', 4, 10))</pre>
<pre># Substring is searched in 's for g'</pre>
<pre>print(word.find('for ', 4, 11))</pre>
Output:
10
-1
-1
6

The Tkinter toolkit comes with all the basic widgets to create graphical applications. Almost every app has a main menu. As expected, Tkinter supports adding a main menu to your application window.

The screenshot below demonstrates a Tkinter based menu:



Related course

[Python Desktop Apps with Tkinter](#)

Tkinter menubar

[You can create a simple menu with Tkinter using the code below.](#)
[Every option \(new, open, save..\) should have its own callback.](#)

```
from Tkinter import *
```

```
def donothing():
```

```
    x = 0
```

```
root = Tk()
```

menubar = Menu(root)	
filemenu = Menu(menubar, tearoff=0)	
filemenu.add_command(label="New", command=donothing)	
filemenu.add_command(label="Open", command=donothing)	
filemenu.add_command(label="Save", command=donothing)	
filemenu.add_separator()	
filemenu.add_command(label="Exit", command=root.quit)	
menubar.add_cascade(label="File", menu=filemenu)	
helpmenu = Menu(menubar, tearoff=0)	
helpmenu.add_command(label="Help Index", command=donothing)	
helpmenu.add_command(label="About...", command=donothing)	
menubar.add_cascade(label="Help", menu=helpmenu)	
root.config(menu=menubar)	
root.mainloop()	
We create the menubar with the call:	
menubar = Menu(root)	
where root is a Tk() object.	
A menubar may contain zero or more submenus such as the file menu, edit menu, view menu, tools menu etcetera.	

A submenu can be created using the same Menu() call, where the first argument is the menubar to attach to.

```
filemenu = Menu(menubar, tearoff=0)
menu = Menu(menubar, tearoff=0)
```

Individual options can be added to these submenus using the add_command() method:

```
filemenu.add_command(label="New",
command=donothing)
filemenu.add_command(label="Open",
command=donothing)
filemenu.add_command(label="Save",
command=donothing)
```

In the example we created the callback function donothing() and linked every command to it for simplicity. An option is added using the add_command() function. We call add_cascade() to add this menu list to the specific list.

[If you are new to programming Tkinter, I highly recommend this course.](#)

[download tkinter examples](#)

The tkinter package is a thin object-oriented layer on top of Tcl/Tk. To use tkinter, you don't need to write Tcl code, but you will need to consult the Tk documentation, and occasionally the Tcl documentation.

[The Tk documentation for tearoff gives you what you're looking for:](#)

tearoff allows you to detach menus for the main window creating floating menus. If you create a menu you will see dotted lines at the top when you click a top menu item. If you click those dotted lines the menu tears off and becomes floating.

[Share](#)

Follow

answered Mar 26 '18 at 18:48

The Tk documentation for `tearoff` gives you what you're looking for:

tearoff allows you to detach menus for the main window creating floating menus. If you create a menu you will see dotted lines at the top when you click a top menu item. If you click those dotted lines the menu tears off and becomes floating.

[Share](#)

Follow

answered Mar 26 '18 at 18:48

[Darrick Herwehe](#)

3,23211 gold badge1818 silver badges2828 bronze badges

What does a floating menu look like? – Stevoisiak Mar 26 '18 at 19:05

@StevenVascellaro Use VLC to see what it looks like, if your OS supports it. – wizzwizz4 Aug 11 '18 at 18:51

[Add a comment](#)

6

Here you can see a tkinter Menu tear-off
I'm not sure how useful this is going to be

Normally, a menu can be torn off: the first choice is occupied by the tear-off element added starting at position 1. If you set the tear-off feature, and choices will be added starting at position 2.

[Share](#)

Follow

[edited Nov 7 '19 at 17:24](#)

answered Mar 26 '18 at 18:53

[Minion Jim](#)

1,03977 silver badges2929 bronze badges

[Add a comment](#)

0

Try this if you want to test the floating menu

```
import tkinter as tk
```

```
root = tk.Tk()
```

```
menubar = tk.Menu(root)
```

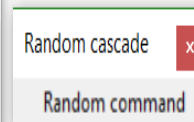
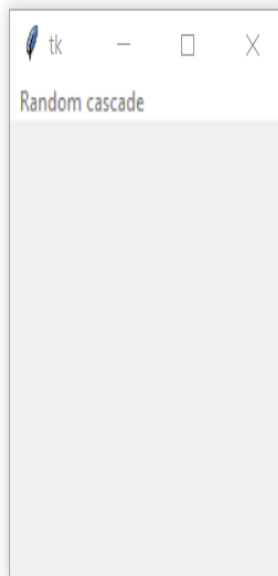
```
rbar = tk.Menu(root)
```

```
rbar.add_command(label = "Random command")
```

```
menubar.add_cascade(label = "Random cascade", menu = rbar)
```

```
root.config(menu = menubar)
```

```
root.mainloop()
```



```
from tkinter import *
```

```
import re
```

```
class HoverInfo(Menu):
```

```
    def __init__(self, parent, text, command=None):
```

```
        self._com = command
```

```
        Menu.__init__(self, parent, tearoff=1)
```

if not isinstance(text, str):
raise TypeError("Trying to initialise a Hover Menu with a non string type: ' + text.__class__.__name__")
toktext=re.split('\n', text)
for t in toktext:
self.add_command(label = t)
self._displayed=False
self.master.bind("<Enter>",self.Display)
self.master.bind("<Leave>",self.Remove)
def __del__(self):
self.master.unbind("<Enter>")
self.master.unbind("<Leave>")
def Display(self,event):
if not self._displayed:
self._displayed=True
self.post(event.x_root, event.y_root)
if self._com != None:
self.master.unbind_all("<Return>")
self.master.bind_all("<Return>", self.Click)
def Remove(self, event):
if self._displayed:
self._displayed=False
self.unpost()
if self._com != None:
self.unbind_all("<Return>")
def Click(self, event):
self._com()

class MyApp(Frame):
def __init__(self, parent=None):
Frame.__init__(self, parent)
self.grid()
self.lbl = Label(self, text='testing')
self.lbl.grid()
self.hover = HoverInfo(self, 'while hovering press return \n for an exciting msg',
self.HelloWorld)
def HelloWorld(self):
print('Hello World')
app = MyApp()
app.master.title('test')
app.mainloop()
This example is Hover Class by Gogo. Display message when hovering over something with mouse cursor in Python
I Just set tear-off to 1 to see the floating effect.
0
By default, the choices in the menu start taking place from position 1. If we set the tearoff = 1, then it will start taking place from 0th position

Layout Managers / Geometry Manager

Introduction

In this chapter of our Python-Tkinter tutorial we will introduce the layout managers or geometry managers, as they are sometimes called as well. Tkinter possess three layout managers:

pack

grid

place

The three layout managers pack, grid, and place should never be mixed in the same master window! Geometry managers serve various functions. They:

arrange widgets on the screen

register widgets with the underlying windowing system

manage the display of widgets on the screen

Arranging widgets on the screen includes determining the size and position of components. Widgets can provide size and alignment information to geometry managers, but the geometry managers has always the final say on the positioning and sizing.

Pack is the easiest to use of the three geometry managers of Tk and Tkinter. Instead of having to declare precisely where a widget should appear on the display screen, we can declare the positions of widgets with the pack command relative to each other. The pack command takes care of the details. Though the pack command is easier to use, this layout managers is limited in its possibilities compared to the grid and place managers. For simple applications it is definitely the manager of choice. For example simple applications like placing a number of widgets side by side, or on top of each other.

Example:

```
import tkinter as tk
```

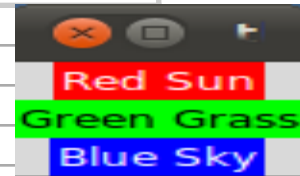
```
root = tk.Tk()
```

```
w = tk.Label(root, text="Red Sun", bg="red", fg="white")
```

```
w.pack()
```

```
w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack()
w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack()

tk.mainloop()
```



In our example, we have packed three labels into the parent widget "root". We used pack() without any options. So pack had to decide which way to arrange the labels. As you can see, it has chosen to place the label widgets on top of each other and centre them. Furthermore, we can see that each label has been given the size of the text. If you want to make the widgets as wide as the parent widget, you have to use the fill=X option:

```
import tkinter as tk

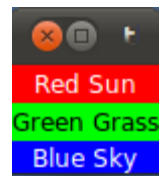
root = tk.Tk()

w = tk.Label(root, text="Red Sun", bg="red", fg="white")
w.pack(fill=tk.X)

w = tk.Label(root, text="Green Grass", bg="green", fg="black")
w.pack(fill=tk.X)

w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")
w.pack(fill=tk.X)

tk.mainloop()
```



The pack() manager knows four padding options, i.e. internal and external padding and padding in x and y direction:

padx

pady

ipadx

ipady

The default value in all cases is 0.

We want to place the three label side by side now and shorten the text slightly:

The corresponding code looks like this:

```
import tkinter as tk
```

```
root = tk.Tk()
```

```
w = tk.Label(root, text="red", bg="red", fg="white")
```

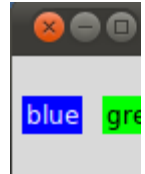
```
w.pack(padx=5, pady=10, side=tk.LEFT)
```

```
w = tk.Label(root, text="green", bg="green", fg="black")
```



```
w.pack(padx=5, pady=20, side=tk.LEFT)
w = tk.Label(root, text="blue", bg="blue", fg="white")
w.pack(padx=5, pady=20, side=tk.LEFT)
tk.mainloop()
```

If we change LEFT to RIGHT in the previous example, we get the colours in reverse order:



The Place geometry manager allows you explicitly set the position and size of a window, either in absolute terms, or relative to another window. The place manager can be accessed through the place method. It can be applied to all standard widgets.

We use the place geometry manager in the following example. We are playing around with colours in this example, i.e. we assign to every label a different colour, which we randomly create using the randrange method of the random module. We calculate the brightness (grey value) of each colour. If the brightness is less than 120, we set the foreground colour (fg) of the label to White otherwise to black, so that the text can be easier read.

```
import tkinter as tk
import random

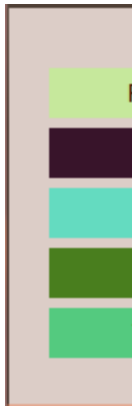
root = tk.Tk()
# width x height + x_offset + y_offset:
root.geometry("170x200+30+30")

languages = ['Python', 'Perl', 'C++', 'Java', 'Tcl/Tk']
labels = range(5)
for i in range(5):
    ct = [random.randrange(256) for x in range(3)]

    brightness = int(round(0.299*ct[0] + 0.587*ct[1] + 0.114*ct[2]))
    ct_hex = "%02x%02x%02x" % tuple(ct)
    bg_colour = '#' + "".join(ct_hex)
    l = tk.Label(root,
                  text=languages[i],

                  fg='White' if brightness < 120 else 'Black',
```





```

        bg=bg_colour)

    l.place(x = 20, y = 30 + i*30, width=120, height=25)

root.mainloop()

The first geometry manager of Tk had been pack. The algorithmic behaviour of
pack is not easy to understand and it can be difficult to change an existing
design. Grid was introduced in 1996 as an alternative to pack. Though grid is
easier to learn and to use and produces nicer layouts, lots of developers keep
using pack.

Grid is in many cases the best choice for general use. While pack is sometimes
not sufficient for changing details in the layout, place gives you complete control
of positioning each element, but this makes it a lot more complex than pack and
grid.

The Grid geometry manager places the widgets in a 2-dimensional table, which
consists of a number of rows and columns. The position of a widget is defined by
a row and a column number. Widgets with the same column number and different
row numbers will be above or below each other. Correspondingly, widgets with
the same row number but different column numbers will be on the same "line"
and will be beside of each other, i.e. to the left or the right.

Using the grid manager means that you create a widget, and use the grid method
to tell the manager in which row and column to place them. The size of the grid
doesn't have to be defined, because the manager automatically determines the
best dimensions for the widgets used.

import tkinter as tk

colours = ['red', 'green', 'orange', 'white', 'yellow', 'blue']

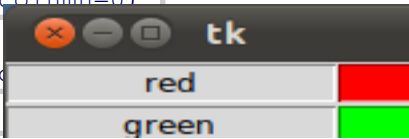
r = 0
for c in colours:





    tk.Label(text=c, relief=tk.RIDGE, width=15).grid(row=r, column=0)

    tk.Entry(bg=c, relief=tk.SUNKEN, width=10).grid(row=r, column=1)

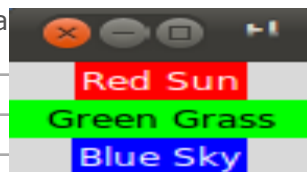
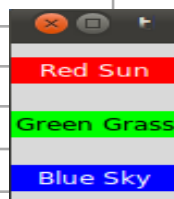
    r = r + 1

```



	-	
	orange	
tk.mainloop()	white	
	yellow	
	blue	

<code>root = tk.Tk()</code>			
<code>w = tk.Label(root, text="Red Sun", bg="red", fg="white")</code>			
<code>w.pack(fill=tk.X, padx=10)</code>			
<code>w = tk.Label(root, text="Green Grass", bg="green", fg="black")</code>			
<code>w.pack(fill=tk.X, padx=10)</code>			
<code>w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")</code>			
<code>w.pack(fill=tk.X, padx=10)</code>			
<code>tk.mainloop()</code>			
External padding, vertically			
The code for the window above:			
<code>import tkinter as tk</code>			
<code>root = tk.Tk()</code>			
<code>w = tk.Label(root, text="Red Sun", bg="red", fg="white")</code>			
<code>w.pack(fill=tk.X, pady=10)</code>			
<code>w = tk.Label(root, text="Green Grass", bg="green", fg="black")</code>			
<code>w.pack(fill= tk.X, pady=10)</code>			
<code>w = tk.Label(root, text="Blue Sky", bg="blue", fg="white")</code>			
<code>w.pack(fill=tk.X, pady=10)</code>			
<code>tk.mainloop()</code>			
Internal padding, horizontally.			
In the following example, we change only the label with the text "Green Grass", so that the result can be easier recognized. We have a			
option.			
<code>import tkinter as tk</code>			
<code>root = tk.Tk()</code>			
<code>w = tk.Label(root, text="Red Sun", bg="red", fg="white")</code>			
<code>w.pack()</code>			
<code>w = tk.Label(root, text="Green Grass", bg="green", fg="black")</code>			



```

w.pack(ipadx=10)
w = tk.Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack()
tk.mainloop()

```

Internal padding, vertically

We will change the last label to ipady=10.



```

import tkinter as tk

root = tk.Tk()

w = tk.Label(root, text="Red Sun", bg="red",
fg="white")
w.pack()

w = tk.Label(root, text="Green Grass",
bg="green", fg="black")
w.pack(ipadx=10)

w = tk.Label(root, text="Blue Sky", bg="blue",
fg="white")
w.pack(ipady=10)
tk.mainloop()

```

