https://www.linuxtuto.com/how-to-install-python-3-12-on-ubuntu-22-04/

https://www.tutorials24x7.com/python/how-to-install-eclipse-for-python-on-ubuntu

Five Data Types

None type – object type that does not contain any value

Numeric types – int, float, complex, bool

Sequences – str, bytes, bytearray, list, tuple, range

Sets

Mapping

type(a) type of variable

d = 3+5j – complex type

int(), float(), bin() for converting different data types

slicing – print(s[0:3])

print(s[::-1])//reverse the string

strip – s.strip(), lstrip(), rstrip() - trim function

find – s.find(“awe”, 0, 10) – 0 is start

count - the occurance of string – s.count(“awe”)

replace – s.replace(old,new)

s.upper(), s.lower(), s.title()

lst = [] array

lst.append(10)

lst.remove(a)

delete by index – del(lst[1])

lst.clear() - clear all elements

lst.insert(3,90) – insert element in specific position

lst.sort()

lst.sort(reverse=True)

Tuple – read only list cannot modify. Immutable

t1 = (1,2,3)

print(t1\*3)

print(t1.count(1))

print(t1.index(1))

----------

Set type – declared in curly braces. It cannot have duplicates

set object does not support indexing, slicing, repitition \*3

f = frozenset(s)

s = {10,20,*"xyz"*,10}

print((s))

s.update([88,90])

print((s))

s.remove(10)

print((s))

------

Range

#r = range(5)

r = range(1,15,3)

for i in r:

print(i)

----

Bytes

lst = [20,30,40,234]

print(type(lst))

b= bytes(lst)

print(type(b))

print((b))

ba = bytearray(lst)

bytearray can be modified.

bytes are immutable

b = bytes([1, 2, 3]) # b[0] = 0 # This will raise a TypeError

---

Dictionary

dict1 = {1:*"aaa"*,2:*"bbb"*}

print(dict1.items())

k = dict1.keys()

for i in k:

print(i)

v = dict1.values()

for i in v:print(i)

print(dict1[2])

del dict1[1]

print(dict1)

----

Immutable – memory location is same for 2 (multiple) variables having same value

------

Constants defined by all caps in variable

–---

a = 10

del a

in python string, integers are immutable. When data types are immutable they point to same memory location

----

Arithmatic operators

a,b = 10,5

print(*'Addition:'*,a+b)

print(*'Substraction:'*,a-b)

print(*'Multi:'*,a\*b)

print(*'Div:'*,a/b)

print(*'Mod:'*,a%b)

print(*'Pow:'*,a\*\*b)

print(*'Floor Div:'*,a//b)performs division and returns integer quotiont

Addition: 15

Substraction: 5

Multi: 50

Div: 2.0

Mod: 0

Pow: 100000

Floor Div: 2

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

Logical Operators – and, or, not

------

Input / output

s=input()

print(s)

name = input(*"Enter your name"*)

print(name)

lst = [x for x in input(*"Enter three numbers separated by space:"*).split()]

print(lst)

print(a,b,sep=’,’)

----------

radius = float(input(*"Enter the radius "*))

pi= 22/7

area = pi\*radius\*\*2

print(*"Area is: "*,area)

----

import math

radius = float(input(*"Enter the radius "*))

#pi= 22/7

area = math.pi\*radius\*\*2

print(*f"Area is: {area:.2f}"*)

-------------------­

if else elif

math,physics,chemistry = [int(x) for x in (input(*"Please enter the marks in math, physics, chemistry:\n "*).split())]

if math<35 or physics<35 or chemistry<35:

print(*"Sorry you are failed in exam"*)

else:

average = (math+physics+chemistry)/3

if average<=59:

print(*"Grade C"*)

elif average<=69:

print(*"Grade B"*)

else:

print(*"grade A"*)

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

While

x=1

while x<=20:

print(x)

x+=1

| **Data Type** | **Characteristics** | **Common Functions/Methods** | **Examples** |
| --- | --- | --- | --- |
| List | - Ordered collection (maintains insertion order). - Mutable (can be changed after creation). - Allows duplicate elements. - Indexed. | - append(), extend(), insert(), remove(), pop(), clear(), index(), count(), sort(), reverse() - len(), min(), max(), sum(), sorted() | - append():  my\_list = [1, 2, 3] my\_list.append(4) # my\_list = [1, 2, 3, 4]  - sort():  my\_list.sort() # my\_list = [1, 2, 3, 4] |
| Tuple | - Ordered collection (maintains insertion order). - Immutable (cannot be changed after creation). - Allows duplicate elements. - Indexed. | - index(), count() - len(), min(), max(), sum(), sorted() | - index():  my\_tuple = (1, 2, 3) my\_tuple.index(2) # returns 1  - count():  my\_tuple.count(2) # returns 1 |
| Set | - Unordered collection. - Mutable (but elements must be immutable). - No duplicate elements allowed. - Unindexed. | - add(), remove(), discard(), pop(), clear(), union(), intersection(), difference(), symmetric\_difference() - len(), min(), max(), sum() | - add():  my\_set = {1, 2, 3} my\_set.add(4) # my\_set = {1, 2, 3, 4}  - union():  set1 = {1, 2} set2 = {2, 3} set1.union(set2) # returns {1, 2, 3} |
| Dictionary | - Unordered collection of key-value pairs. - Mutable. - Keys must be unique and immutable. - Values can be of any data type. - Unindexed. | - get(), keys(), values(), items(), pop(), popitem(), update(), clear(), setdefault() - len(), min(), max(), sum() | - get():  my\_dict = {'a': 1, 'b': 2} my\_dict.get('a') # returns 1  - items():  my\_dict.items() # returns dict\_items([('a', 1), ('b', 2)]) |
| String | - Ordered sequence of characters. - Immutable. - Indexed. - Allows duplicate characters. | - upper(), lower(), strip(), replace(), split(), join(), find(), count(), format(), startswith(), endswith() - len(), min(), max(), sorted() | - upper():  my\_str = "hello" my\_str.upper() # returns 'HELLO'  - split():  my\_str = "hello world" my\_str.split(' ') # returns ['hello', 'world'] |

### Summary of Data Types:

1. List: A flexible, ordered collection that can hold any data type. It's mutable, so you can modify it after creation. Lists are used when you need an ordered, indexable collection that can grow or shrink.
2. Tuple: Similar to a list, but immutable. Once created, you cannot change the elements of a tuple. Tuples are used when you need an ordered, indexable collection that should not change.
3. Set: An unordered collection of unique elements. Useful when you need to store distinct items and perform operations like union, intersection, and difference.
4. Dictionary: An unordered collection of key-value pairs. Keys must be unique and are used to access the corresponding values. Dictionaries are used when you need a collection that maps unique keys to values.
5. String: A sequence of characters, immutable and indexed. Strings are used for text manipulation and storing character data.

---

for i in range(50,71,2):

print(i)

-----

num = int(input(*"Enter a number"*))

*"""lst = [1,2,3,4,5,6,7,8,9,10]*

*table = 0*

*for i in lst:*

*table+=num*

*print(table)"""*

for i in range(1,11):

print(num,*'X'*,i,*'='*,i\*num)

----

l1 = eval(input(*"Please enter list of elements "*))

print(l1)

l2 = []

for each\_value in l1:

if(each\_value not in l2):

l2.append(each\_value)

print(l2)

----

l1 = eval(input(*"Please enter list of elements "*))

print(l1)

l3 = set(l1)

print(l3)

l2 = []

for each\_value in l1:

if(each\_value not in l2):

l2.append(each\_value)

print(l2)

----

vowels

word = input(*"Please enter any word: "*)

result = {}

vowels = {*"a"*, *"e"*, *"i"*, *"o"*, *"u"*}

for c in word:

if c in vowels:

result[c] = result.get(c,0)+1

for k,v in result.items():

print(k, *"present "*, v, *"times"*)

-----

Employee details

The get() method in a dictionary is used to retrieve the value associated with a specified key. If the key is not found in the dictionary, it returns None by default or a specified default value if provided as a second parameter.

n = int(input(*"Enter employee number "*))

employees = {}

for i in range(n):

name = input(*"Enter employee name"*)

salary = input(*"Enter salary"*)

employees[name] = salary

while True:

name = input(*"Enter employee name"*)

salary = employees.get(name,-1)

if(salary==-1):

print(*"Employee does not exist"*)

else:

print(*"Salary of employee is"*, salary)

choice = input(*"Do you want to continue? y or n"*)

if(choice==*"n"*):

break

----------

reverse a string

slicing ::-1

or using len() function, while loop,i = i-1

----

join method for reverse a string

s = *"-"*.join([*'a'*,*'b'*,*'c'*])

print(s)

a-b-c

st = input(*"Enter a string"*)

print(*''*.join(reversed(st)))

reversed return a object

-----

split method for reverse the words

---

count the number of characters

s = *"ddfdfhhhkkhhhhh"*

d = {}

for c in s:

if(c in d.keys()):

d[c] = d[c] + 1

else:

d[c] = 1

for k,v in d.items():

print(*'{}={} times '*.format(k,v))

-----------

for pattern use this

print(*"\*"*,end=*""*) or use print(“\*”\*i) in for loop range

pyramid pattern -

for I in range(1,n+1):

print(“ “\*(n-I),end=””)

print(“\*”\*i)

---

find substr in str

pos = s.find(sub,pos+1,length) this is in while loop

--

====

command line

---

Accessing global variables

print(globals()[‘x’])

---

assigning function to a variables

def display():

print(“run”)

z = display

---

def **display**(fun):

return *"hellp "*+fun

def **name**():

return *"Shreyas"*

print(display(name()))

----

recursion

def factorial(n)

if n==0:

return 1

else:

return n\*factorial(n-1)

-----

keyword argument

print(average(b=10,a=20))

---

\*args optional parameters

\*\*kwargs – arguments with keywords

Command can be used like python3 calc.py subtract 5 3

OR

python3 calc.py sum 5 3

----

Lambda – Anonymous function

def square(x):

return x\*x

function = lambda x:x\*x

f = lambda n:n\*\*3

print(f(2))

----------

filter function

lst = [10,3,5,34,67,66]

fn = list(filter(lambda x:x%2==0,lst))

print(fn)

---

reduce

from functools import reduce

lst = [3,4,5]

fn = reduce(lambda x,y:x+y, lst)

print(fn)

---

map

lst = [3,4,5]

fn = list(map(lambda x:x\*2,lst))

print(fn)

------

decorator

def **decor**(fun):

def **inner**():

result = fun()

return result\*2

return inner

@decor

def **num**():

return 5

resultfun = decor(num)

print(resultfun())

décor function take function as parameter and returns a function

----------

generator

def **custom**(x,y):

while x<y:

yield x

x+=1

result = custom(20,30)

#print(result)

for i in result:

print(i)

------

Module – group of functions

import mymath1

print(mymath1.add(10,11))

OR  
from mymath1 import \*

print(add(10,11))

---

from math import \*

print(sqrt(8))

print(ceil(8.2))

print(floor(9.6))

print(fabs(-2.2))

print(dir())

print(dir(math))

help(math)

Random module

from random import \*

#for i in range(10):

# print(random())

#for i in range(10):

# print(randint(1,50))

#for i in range(10):

# print(uniform(1,50))

#for i in range(10):

# print(randrange(1,12))

lst = [*"apple"*,*"guava"*,*"mango"*,*"jelly"*]

for i in range(10):

print(choice(lst))

–

from random import \*

def **number\_guessing\_game**():

num = randint(1,100)

print(*"Welcome to Number Guessing Game"*)

guess = None

while guess!=num:

guess = int(input(*"Enter your guess: "*))

if guess<num:

print(*"Too low. Try again!"*)

elif guess>num:

print(*"Too high .. try again!"*)

else:

print(*"Congrats.."*)

number\_guessing\_game()

----

list comprehension

product of two list and get common values

lst1 = [2,0,3,4,5]

lst2 = [10,20,5,30,0]

lst3 = []

*'''for i in range(len(lst1)):*

*lst3.append(lst1[i]\*lst2[i])'''*

lst3 = [lst1[i]\*lst2[i] for i in range(len(lst1))]

*'''for i in range(len(lst1)):*

*if lst1[i] in lst2:*

*lst3.append(lst1[i])'''*

lst3 = [lst1[i] for i in range(len(lst1)) if lst1[i] in lst2]

lst3 = [i for i in lst1 if i in lst2]

print(lst3)

-----------

OOPS

classes – collection of methods and variables. Blueprint for an object.

class **Product**:

def **\_\_init\_\_**(*self*):

*self*.name = *"iphone"*

*self*.description = *"This is awesome"*

*self*.price = 40000

p1 = Product()

print(p1.description)

print(p1.name)

print(p1.price)

---

get and set

class **Programmer**:

def **setName**(*self*,n):

*self*.name = n

def **getName**(*self*):

return *self*.name

def **setSalary**(*self*,s):

*self*.salary = s

def **getSalary**(*self*):

return *self*.salary

def **setTech**(*self*,t):

*self*.tech = t

def **getTech**(*self*):

return *self*.tech

p1 = Programmer()

p1.setName(*"Shreyas"*)

p1.setSalary(3000000)

p1.setTech(*"python"*)

print(p1.getName())

print(p1.getSalary())

print(p1.getTech())

----

static variables and instance variables

class **Student**:

major = *"Gem"*

def **\_\_init\_\_**(*self*,s):

*self*.name=s

---

Garbage Collection

def \_\_del\_\_(self) - desctructor

import gc

gc.disabled()

gc.isenabled()

p1 = None

–

encapsulation

class **Student**:

def **\_\_init\_\_**(*self*,s):

*self*.\_\_name=s (two underscore)

create display method to access private variables OR

print(s.student\_\_name) object . underscore classname underscore underscore name

-------

Inheritance

class **BMW**:

def **\_\_init\_\_**(*self*,make,model,year):

*self*.make = make

*self*.model = model

*self*.year = year

def **start**(*self*):

print(*"Starting the car"*)

def **stop**(*self*):

print(*"Stopping the car"*)

class **ThreeSeries**(BMW):

def **\_\_init\_\_**(*self*,cruiseControlEnabled,make,model,year):

#BMW.\_\_init\_\_(self, make, model, year)

super().\_\_init\_\_( make, model, year)

*self*.cruiseControlEnabled = cruiseControlEnabled

def **display**(*self*):

print(*self*.cruiseControlEnabled)

def **start**(*self*):

super().start()

print(*"button start"*)

class **FiveSeries**(BMW):

def **\_\_init\_\_**(*self*,parkingAssistEnabled,make,model,year):

BMW.\_\_init\_\_(*self*, make, model, year)

*self*.parkingAssistEnabled = parkingAssistEnabled

threeSeries = ThreeSeries(True,*"BMW"*,*"329i"*,*"2018"*)

print(threeSeries.cruiseControlEnabled)

print(threeSeries.make)

print(threeSeries.model)

print(threeSeries.year)

threeSeries.start()

threeSeries.display()

----

ducktyping – same function acts different when we pass different objs dynamically

class **Flight**:

def **\_\_init\_\_**(*self*,engine):

*self*.engine = engine

def **startEngine**(*self*):

*self*.engine.start()

class **AirBusEngine**:

def **start**(*self*):

print(*"Starting Air Bus Engine"*)

class **BoingEngine**:

def **start**(*self*):

print(*"Start Boing ENgine"*)

ae = AirBusEngine()

f = Flight(ae)

f.startEngine()

+ operator is overloaded and polymorphic

Run time polymorphism comes free due to dynamic typing.

Abstraction

Abstract method declared in parent class then it must be written in child class too

from abc import abstractmethod,ABC

class **BMW**(ABC):

def **\_\_init\_\_**(*self*,make,model,year):

*self*.make = make

*self*.model = model

*self*.year = year

def **start**(*self*):

print(*"Starting the car"*)

def **stop**(*self*):

print(*"Stopping the car"*)

*@abstractmethod*

def **drive**(*self*):

pass

class **ThreeSeries**(BMW):

def **\_\_init\_\_**(*self*,cruiseControlEnabled,make,model,year):

#BMW.\_\_init\_\_(self, make, model, year)

super().\_\_init\_\_( make, model, year)

*self*.cruiseControlEnabled = cruiseControlEnabled

def **display**(*self*):

print(*self*.cruiseControlEnabled)

def **start**(*self*):

super().start()

print(*"button start"*)

def **drive**(*self*):

print(*"in drive"*)

class **FiveSeries**(BMW):

def **\_\_init\_\_**(*self*,parkingAssistEnabled,make,model,year):

BMW.\_\_init\_\_(*self*, make, model, year)

*self*.parkingAssistEnabled = parkingAssistEnabled

def **drive**(*self*):

print(*"in drive"*)

Interface is class where all the methods are abstract methods

overriding in object-oriented programming – runtime polymorphism

Exception

try:

lst = [10,20,30,40,50,60]

i = int(input(*f"Enter index between 0 {len(lst)-1}"*))

print(*"Value in that index is:"*,lst[i])

except IndexError:

print(*"value beyond index"*)

IndexError, ZeroDivisionError

Finally

try:

f = open(*"file"*,*"w"*)

a,b = [int(x) for x in input(*"Enter two numbers"*).split()]

c = a/b

f.write(*"Writing %d into file"* %c)

print(c)

except ZeroDivisionError:

print(*"zero divisible not allowed"*)

else:

print(*"Numbers are correct"*)

finally:

f.close()

print(*"file close"*)

print(*"code after exception"*)

----

class **TooYoungException**(Exception):

def **\_\_init\_\_**(*self*,msg):

*self*.msg = msg

class **TooOldException**(Exception):

def **\_\_init\_\_**(*self*,msg):

*self*.msg = msg

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

if age<18:

raise TooYoungException(*"You are too young"*)

elif age>90:

raise TooOldException(*"Too old"*)

else:

print(*"Eligible"*)

-----

Logging

import logging

logging.basicConfig(filename=*"mylog.log"*,level=logging.DEBUG)

logging.critical(*"Critical"*)

logging.error(*"Error"*)

logging.warning(*"Warning"*)

logging.info(*"Info"*)

logging.debug(*"Debug"*)

–

import logging

logging.basicConfig(filename=*"mylog.log"*,level=logging.ERROR)

try:

f = open(*"file"*,*"w"*)

a,b = [int(x) for x in input(*"Enter two numbers"*).split()]

c = a/b

f.write(*"Writing %d into file"* %c)

print(c)

except ZeroDivisionError:

print(*"zero divisible not allowed"*)

logging.error(*"This islog"*)

else:

print(*"Numbers are correct"*)

finally:

f.close()

print(*"file close"*)

print(*"code after exception"*)

–

Assert

try:

num = int(input(*"Please enter even number "*))

assert num%2==0, *"You have entered invalid number"*

except AssertionError as obj:

print(obj)

print(*"After the assertion"*)

---

File

f = open(*"myfile.txt"*,*"w"*)

s = input(*"Enter Text:"*)

f.write(s)

f.close()

–

f.seek(0)

print(f.readline()) //12 parameter ...a number

print(f.readlines())

f.writelines([]) f.seek() is required

cursor at which point – f.tell()

a+ a= append

for line in f:

a.append(line)

print(a)

–

import os,sys

if(os.path.isfile(*"myfile.txt"*)):

f = open(*"myfile.txt"*,*"r"*)

else:

print(*"File does not exists"*)

sys.exit()

s = f.read()

print(s)

f.close()

---

regular expression

\d – digit

\D – non digit

\s – space

\S – non space

\w – alpha numberic

\W – non alpha

\A – beginning search

\Z – end search

import re

result = re.search(r’o\w\w’,str) starts with o alphanum

result.group()

result = re.findall(r’o\w\w’,str)

result

result = re.match(r’o\w\w’,str) match only one ..first

result.group()

re.sub(r’one’,’two’,str) replace with second str

re.split(r’\d+’,str)

quantifiers = + one or more

\* - zero or more

{m}

{m,n} minimum m , maximum n

? = 0 or one

find date

re.findall(r’\d{1,2}-\d{1,2}-\d{4},str)

\ - escape characters

. - newline

^ - beginning of a string

$ - end of string

[…] range

[^…] except these

(r|s) multiple regex

result = re.search(r’^T\w’,str)

import re

import urllib.request – get url details

----

date time

import time, datetime

epoch – time.time()

time.ctime(epoch)

dt – datetime.datetime.today()

dt.day dt.month etc

from datetime import \*

d = date(2018,7,18)

t = time(12,34)

dt = datetime.combine(d,t)

–

ldate = []

ldate(append(date(2027,1,6))

ldate.sort()

–

import time

time.sleep(3) wait 3 sec

time.perf\_counter() - performance measure. Put this on start and end

---

datetime.now().date()

---

all programs we did were single threaded

import threading

print(*"Current running thread "*,threading.current\_thread().getName())

Create thread using functions

from threading import Thread

def **displayNumber**():

i=0

while(i<10):

print(i)

i+=1

t = Thread(target=displayNumber)

t.start()

---

from threading import \*

def **displayNumber**():

i=0

print(current\_thread().getName())

while(i<10):

print(i)

i+=1

print(current\_thread().getName())

t = Thread(target=displayNumber)

t.start()

---

Thread extending thread class

from threading import Thread

class **MyThread**(Thread):

def **run**(*self*):

i=0

while(i<10):

print(i)

i+=1

t = MyThread()

t.start()

–

Thread using class

from threading import \*

class **MyT**:

def **run**(*self*):

i=0

print(current\_thread().getName())

sleep(1)

while(i<10):

print(i)

i+=1

obj = MyT()

t = Thread(target=obj.run)

t.start()

t1 = Thread(target=obj.run)

t1.start()

t2 = Thread(target=obj.run)

t2.start()

locking the code for particular thread

from threading import \*

class **Mybus**:

def **\_\_init\_\_**(*self*,availableseats):

*self*.availableseats = availableseats

*self*.l = Lock()

def **buy**(*self*,requestedseats):

*self*.l.acquire()

print(*"total seats:"*,*self*.availableseats)

if(requestedseats<*self*.availableseats):

print(*"Booking ticket"*)

print(*"Making Payment"*)

print(*"Printing ticket"*)

*self*.availableseats -= requestedseats

else:

print(*"Sorry no seats"*)

*self*.l.release()

obj = Mybus(10)

t = Thread(target=obj.buy,args=(3,))

t1 = Thread(target=obj.buy,args=(3,))

t2 = Thread(target=obj.buy,args=(3,))

t.start()

t1.start()

t2.start()

-----

Boolean Flag

from threading import \*

from time import \*

class **producer**:

def **\_\_init\_\_**(*self*):

*self*.products = []

*self*.orderedplaced = False

def **produce**(*self*):

for i in range(1,5):

*self*.products.append(*"product"*+str(i))

sleep(1)

print(*"Item added"*)

*self*.orderedplaced = True

class **Consumer**:

def **\_\_init\_\_**(*self*,prod):

*self*.prod = prod

def **consume**(*self*):

while *self*.prod.orderedplaced == False:

sleep(0.2)

print(*"Shipped"*,*self*.prod.products)

P = producer()

C = Consumer(P)

t1 = Thread(target=P.produce)

t2 = Thread(target=C.consume)

t1.start()

t2.start()

-----

wait and notify – lock is to keep threads synchronise

from threading import \*

from time import \*

class **producer**:

def **\_\_init\_\_**(*self*):

*self*.products = []

*self*.c = Condition()

def **produce**(*self*):

*self*.c.acquire()

for i in range(1,5):

*self*.products.append(*"product"*+str(i))

sleep(1)

print(*"Item added"*)

*self*.c.notify()

*self*.c.release()

class **Consumer**:

def **\_\_init\_\_**(*self*,prod):

*self*.prod = prod

def **consume**(*self*):

*self*.prod.c.acquire()

*self*.prod.c.wait(timeout=0)

*self*.prod.c.release()

print(*"Shipped"*,*self*.prod.products)

P = producer()

C = Consumer(P)

t1 = Thread(target=P.produce)

t2 = Thread(target=C.consume)

t1.start()

t2.start()

-----

Queue Producer consumer pattern using queue

import random

import time

import queue

from threading import \*

def **producer**(q):

while True:

print(*"producing"*)

q.put(random.randint(1,50))

print(*"Produced"*)

time.sleep(3)

def **consumer**(q):

while True:

print(*"Ready to consume"*)

print(*"Consumedata"*,q.get())

time.sleep(3)

q = queue.Queue()

t1 = Thread(target=consumer,args=(q,))

t2 = Thread(target=producer,args=(q,))

t1.start()

t2.start()

----

Queue types

import queue

q= queue.Queue()

q.put(100)

q.put(200)

q.put(400)

while not q.empty():

print(q.get(),end=*' '*)

q= queue.LifoQueue() - reverse the order

q= queue.PriorityQueue() - ascending order

-----

Networking

import urllib.request

try:

url= urllib.request.urlopen(*"https://python.org"*)

content = url.read()

url.close()

except urllib.error.HTTPError:

print(*"Web page is not found"*)

f = open(*"python.html"*,*"wb"*)

f.write(content)

f.close()

image

import urllib.request

url = *"https://timemaster.creatingwow.in/assets/TM%20logo.svg"*

urllib.request.urlretrieve(url, *"ib.svg"*)

----

Socket Programming

Server Connect

import socket

host = *'localhost'*

port = 4000

s= socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.bind((host,port))

print(*"Server is listening on port:"*,port)

s.listen(1)

c,addr = s.accept()

print(*"Connection from:"*,str(addr))

c.send(*b"Hello How are you?"*)

c.send(*"bye"*.encode())

---

Client connect

import socket

s = socket.socket()

s.connect((*'localhost'*,4000))

msg = s.recv(1024)

while msg:

print(*"Received:"*, msg.decode())

msg = s.recv(1024)

---

Get a file client

import socket

s = socket.socket()

s.connect((*'localhost'*,4000))

fileName = input(*"Enter a filename"*)

s.send(fileName.encode())

content = s.recv(1024)

print(content.decode())

s.close()

---

krvt tndn oinz aiso

---

send email

import smtplib

from email.mime.text import MIMEText

body = *"This is the test email from python tutorial"*

msg = MIMEText(body)

msg[*'From'*] = *"shreyas.dhokte@infobeans.com"*

msg[*'To'*] = *"shreyas.dhokte@infobeans.com"*

msg[*'Subject'*] = *"Hello"*

server = smtplib.SMTP(*'smtp.gmail.com'*,587)

server.starttls()

server.login(*"shreyas.dhokte@infobeans.com"*,*'krvt tndn oinz aiso'*)

server.send\_message(msg)

print(*"mail sent"*)

server.quit()

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

Databases

pip3 install mysql-connector-python

import mysql.connector

conn = mysql.connector.connect(host=*'localhost'*,database=*'pythondb'*,username=*'root'*,password=*'Root@123'*)

if conn.is\_connected():

print(*"Connected"*)

cursor = conn.cursor()

cursor.execute(*'select \* from emp'*)

*"""row = cursor.fetchone()*

*while row is not None:*

*print(row)*

*row = cursor.fetchone()"""*

rows = cursor.fetchall()

print(*"Total number of records"*,cursor.rowcount)

for row in rows:

print(row)

conn.close()

Finally block executes wherever exception is or not

=---

create

import mysql.connector

conn = mysql.connector.connect(host=*'localhost'*,database=*'pythondb'*,username=*'root'*,password=*'Root@123'*)

if conn.is\_connected():

print(*"Connected"*)

cursor = conn.cursor()

try:

cursor.execute(*"INSERT INTO `emp`(`idemp`,`name`,`salary`)VALUES(300,'Abby',5000)"*)

conn.commit()

except:

conn.rollback()

conn.close()

---

delete

---

Postgres

sudo -u postgres psql

----

Mongodb is unstructure db, json format

sudo systemctl enable mongod

sudo systemctl stop mongod

sudo systemctl stop mongod

pip3 install pymongo

from pymongo import MongoClient

client = MongoClient(*'localhost'*,27017)

database = client[*'mydb'*]

print(*"Database created"*)

collection = database[*'product'*]

print(*"collection created"*)

products = [{

*"name"*:*"Iphone"*,

*"price"*:1200

},

{

*"name"*:*"samsung"*,

*"price"*:400

},

{

*"name"*:*"google"*,

*"price"*:500

}]

result = collection.insert\_many(products)

print(result.inserted\_ids)

----

update

from pymongo import MongoClient

client = MongoClient(*'localhost'*,27017)

database = client[*'mydb'*]

collection = database[*'product'*]

print(*"Database created"*)

filter = {*"name"*:*"samsung"*}

collection.update\_one(filter,{*"$set"*:{*"price"*:3}})

cursor = collection.find({*"name"*:*"samsung"*})

for each in cursor:

print(each)

delete

from pymongo import MongoClient

client = MongoClient(*'localhost'*,27017)

database = client[*'mydb'*]

collection = database[*'product'*]

collection.delete\_one({*"name"*:*"iphone"*})

cursor = collection.find({*"name"*:*"iphone"*})

for each in cursor:

print(each)

----

pip3 – packaging manager

python3 -m venv ./venv

which pip3

source ./venv/bin/activate

then pip3 install django

Unit Test

import unittest

from cc import \*

class **Credit**(unittest.TestCase):

def **test\_ValidateCard\_valid**(*self*):

result = ValidateCard(date(2025,2,2))

*self*.assertEqual(*'Valid'*, result)

if \_\_name\_\_ == *'\_\_main\_\_'*:

unittest.main()

---

numpy

from numpy import \*

arr = array([1,2,3])

print(arr)

---

python3 -m venv venv\_name

source venv\_name/bin/activate

pip freeze

–

source env\_jango/bin/activate

pip install django==4.2

django-admin –version

pip freeze

–

django-admin startproject crudexample

cd crudexample

python3 manage.py makemigrations

python3 manage.py migrate

python3 manage.py createsuperuser

python3 manage.py startapp employee

username admin / admin

python3 manage.py startapp employee

python3 manage.py shell

9545372959 – rishabh raj

https://drive.google.com/drive/folders/1cVo0OzZDbea1hG-GOBqXVQ3CknG5wWe-

source env\_jango/bin/activate

python3 manage.py runserver

cd path/to/load\_forecast\_tool

create virtual environment

python3 -m venv env

source env/bin/activate

pip install -r requirements.txt

python3 manage.py migrate

python3 manage.py createsuperuser

pip install djangorestframework

–----

port - python3 manage.py runserver 4444

–

views are function base and class base

----

create project

django-admin startproject firstproject

---

run project

python3 manage.py runserver

---

create app

python3 manage.py startapp firstApp

---

in settings.py add appname in the list

ctrl shift p then python:select interpreter

---

create virtual evironment

python3 -m venv env

source env/bin/activate

then

pip install django==4.2

then

sudo apt-get install python3-dev default-libmysqlclient-dev build-essential

then

pip install mysqlclient

start project

env/bin/django-admin startproject VirtualEnvDemo

----

python3 manage.py migrate

python3 manage.py runserver

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

python3 manage.py shell

from django.db import connection

c = connection.cursor()

–-----

python3 manage.py makemigrations

python3 manage.py migrate

-----

'DIRS': [BASE\_DIR / 'templates'],

STATIC\_URL = 'static/'

STATICFILES\_DIRS = [BASE\_DIR / 'static']

-------

python3 manage.py createsuperuser

------

admin.site.register(Employee)

---------

Django ORM

python3 manage.py shell

from empApp.models import Employee

to get multiple records – Employee.objects.all()

to get single record – Employee.objects.get(id=1)

WHERE

emps = Employee.objects.filter(salary\_\_gt=5000)

gt, gte, lt, lte, contains, icontains in startswith endswith

---

from django.db.model import Q

emps = Employee.objects.filter(Q(firstName\_\_startswith=”bh”) | Q(lastName\_\_startswith=”xy”))

same for &

emps = Employee.objects.filter(firstName\_\_startswith=”bh”,lastName\_\_startswith=”xy”)

exclude

Employee.objects.exclude(salary\_\_gt=5000)

----

only specific columns

emp = Employee.objects.all().values\_list(‘firstName’,’salary’)

emp = Employee.objects.all().values(‘firstName’,’salary’)

emp = Employee.objects.all().only(‘firstName’,’salary’)

-----

from django.db.models import Avg,Sum,Max,Min,Count

emp = Employee.objects.all().aggregate(Avg(‘salary’))

----

Create or insert new record

emp = Employee.objects.create(firstName=’bob’,lastName=’ser’,salary=’4000’)

count = Employee.objects.count()

-----

Create bulk records

Employee.objects.bulk\_create([Employee(firstName=’Jeff’,lastName=’ff’),Employee(firstName=’Jeff’,lastName=’ff’)])

Passenger.objects.bulk\_create([Passenger(firstName=’Jeff’,lastName=’ff’,email=’ss@ss.com’),Employee(firstName=’James’,lastName=’ff’)])

---

Delete

e = Employee.objects.get(id=1)

e.delete()

----

bulk delete

qs = Employee.objects.filter(salary\_\_gt=5000)

qs.count()

qs.delete()

qs = Employee.objects.all().delete()

-----------

Update the record

emp = Employee.objects.get(id=2)

emp.firstName = “newname”

emp.save()

emps = Employee.objects.all().order\_by(‘salary’)

emps = Employee.objects.all().order\_by(‘-salary’)

emps = Employee.objects.all().order\_by(‘salary’)[0]

emps = Employee.objects.all().order\_by(‘salary’)[0:3]

-------

form

{{form.as\_p}}