1. Ans=6. At the first line, a is being initialized as a global variable. Function b refers to that same global viable a; and calls function c so as to assign a value to it. Function c takes up the value as local variable a, and returns as (a+2). This value is being thus assigned to the global variable a.  
   Upon calling b once, the function initializes a as the global variable, and by calling function c, changes the value of a to 2.  
   Upon calling b again, the same process is repeated, and the value of a is updated from 2 to 4.  
   Upon calling b again, the same process is repeated again, and the value of a is updated to 6.
2. #Snippet\_on\_exception\_handling:

def file\_length(file\_name):

try:

file = open(file\_name)

contents = file.read()

file.close()

print(len(contents))

except FileNotFoundError:

print("File "+file\_name+" not Found.")

1. #Snippet\_on\_inheritance:

class Marsupial:

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

self.pouch = list()

def put\_in\_pouch(self, item):

self.pouch.append(item)

def pouch\_contents(self):

print(self.pouch)

class Kangaroo(Marsupial):

def \_\_init\_\_(self, x, y):

super().\_\_init\_\_()

self.x = x

self.y = y

def jump(self, dx, dy):

self.x += dx

self.y += dy

def \_\_str\_\_(self):

return f"I am a Kangaroo located at coordinates ({self.x},{self.y})"

1. #Snippet\_on\_recursion:

def collatz(x):

print(x)

if x == 1:

return x

elif x % 2 == 0:

collatz(x//2)

else:

collatz(3\*x+1)

1. #Snippet\_on\_recursive\_binary

def binary(n):

if n<0:

print("No negative number allowed.")

return -1

if n > 1:

binary(n // 2)

print(n % 2, end='')

1. #Snippet\_on\_ HTMLParser

from html.parser import HTMLParser

class HeadingParser(HTMLParser):

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

super().\_\_init\_\_()

self.indent = 0

self.inside\_body = False

def handle\_starttag(self, tag, attrs):

if tag == 'body':

self.inside\_body = True

elif tag in ['h1', 'h2', 'h3', 'h4', 'h5', 'h6'] and self.inside\_body:

self.indent = int(tag[1]) - 1

def endtagHandler(self, tag):

if tag in ['h1', 'h2', 'h3', 'h4', 'h5', 'h6'] and self.inside\_body:

self.indent = 0

elif tag == 'body':

self.inside\_body = False

def handle\_data(self, data):

if self.inside\_body and data.strip() != '':

print((' ' \* self.indent) + data.strip())

1. #Snippet1\_on\_urlRequests:

import re, urllib.request

def webdir(url, depth, indent):

# Print the URL of the current page with appropriate indentation

print(" " \* indent + url)

if depth == 0:

return

try:

with urllib.request.urlopen(url) as f:

content = f.read().decode('utf-8')

links = re.findall(r'<a\s+(?:[^>]\*?\s+)?href="([^"]\*)"', content)

if not links:

print(" " \* (indent + 2) + "No links found on this page")

return

for link in links:

# Resolve relative links to absolute URLs

if not link.startswith("http"):

link = urllib.parse.urljoin(url, link)

# Recursively visit each link

webdir(link, depth - 1, indent + 2)

except:

pass

1. //SQL\_queries
2. *All the temperature data.*

SELECT Temperature FROM weather;

1. *All the cities, but without repetition.*

SELECT DISTINCT City FROM weather;

1. *All the records for India.*

SELECT \* FROM weather WHERE Country = 'India';

1. *All the Fall records.*

SELECT \* FROM weather WHERE Season = 'Fall';

1. *The city, country, and season for which the average rainfall is between 200 and 400 millimeters.*

SELECT City, Country, Season FROM weather WHERE Rainfall BETWEEN 200 AND 400 GROUP BY City, Country, Season;

1. *The city and country for which the average Fall temperature is above 20 degrees, in increasing temperature order.*

SELECT City, Country FROM weather WHERE Season = 'Fall' GROUP BY City, Country HAVING AVG(Temperature) > 20 ORDER BY AVG(Temperature) ASC;

1. *The total annual rainfall for Cairo.*

SELECT SUM(Rainfall) AS TotalRainfall FROM weather WHERE City = 'Cairo';

1. *The total rainfall for each season.*

SELECT Season, SUM(Rainfall) AS TotalRainfall FROM weather GROUP BY Season;

1. #list\_comprehension

[word.upper() for word in words]

[word.lower() for word in words]

[len(word) for word in words]

[[word.upper(), word.lower(), len(word)] for word in words]

[word for word in words if len(word) >= 4]