```
num_list=[2,3,5,7,8,9,12,16]
for num in num_list:
    if(num*num)%8==0:
        print(num)

->1b

if(amount>2999):
    print("you are eligible for purchase")    // syntaxerrror
    if(amount>2999):


print("Hello, World!"  // syntax error of missing closing parenthesis

# Value Error
num = int("abc")


 # Type Error
result = "Hello" + 5

# Index Error
my_list = [1, 2, 3]
print(my_list[5])

->pgm 2

input_string="python is high level,general purpose programming language"
print("the input string is:",input_string)
myset=set(input_string)

for element in myset:
    countOfChar=0
    for character in input_string:
        if character==element:
            countOfChar+=1
    print("count of character '{}' is {}".format(element,countOfChar))


-> pgm 3

agencies = {
    "CBI": "Central Bureau of Investigation",
    "FBI": "Federal Bureau of Investigation",
    "NIA": "National Investigation Agency",
    "SSB": "Service Selection Board",
    "WPA": "Works Progress Administration"
}

print(agencies)
print("******")
print(type(agencies))

# Adding the new entry for BSE
```

```
agencies["BSE"] = "Bombay Stock Exchange"

print(agencies)
```

->pgm 4

```
import turtle
num_sides = int(input("Enter the number of sides: "))
side_length = int(input("Enter the length of each side: "))
pen_color = input("Enter the pen color: ")
fill_color = input("Enter the fill color: ")
# Set
t = turtle.Turtle()
t.color(pen_color)
t.fillcolor(fill_color)
# Draw polygon
angle = 360 / num_sides
t.begin_fill()
for i in range(num_sides):
    t.forward(side_length)
    t.right(angle)
t.end_fill()
# Keep turtle window open until user clicks to close
turtle.done()
```

->PGM 5

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]


        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

arr = [12, 11, 13, 5, 6, 7]

merge_sort(arr)

print("Sorted array is:", arr)
```

->PGM 5B

```
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

arr = [2, 3, 4, 10, 40]
target = 10

result = binary_search(arr, target)

if result != -1:
    print("Element is present at index", result)
else:
    print("Element is not present in array")
```

->PGM 6

```
def pay(time, wage):
    if time>60:
       return 2*time*wage
    elif time>40:
      return 1.5*time*wage
    else:
      return time*wage
time = int(input("Enter the hours worked in last week:"))
wage = float(input("Enter wage per hour:"))
print("Your's week pay is:", pay(time, wage))
```

->PGM 7

```
class BankAccount:
    def __init__(self, balance=0):
        self.balance = balance
    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount
            print(f"{amount} withdrawn successfully.")
        else:
            print("Not enough balance.")

    def deposit(self, amount):
        self.balance += amount
        print(f"{amount} successfully deposited.")

    def show_balance(self):
        print(f"The balance is {self.balance}")


account = BankAccount(int(input("Enter the opening balance: ")))

while True:
    print("\nBank Account Operations")
    print("1. Withdraw\n2. Deposit\n3. Check Balance\n4. Exit")

    option = int(input("Choose an option: "))

    if option == 1:
        account.withdraw(int(input("Enter the amount to withdraw: ")))
    elif option == 2:
        account.deposit(int(input("Enter the amount to deposit: ")))
    elif option == 3:
        account.show_balance()
    elif option == 4:
        print("Exiting...")
        break
    else:
        print("Invalid option. Please choose again.")
```

->PGM 8

```
print("Welcome To Bike Shop")

bikes = ["MTB", "Geared", "Non-Geared", "With Training Wheels", "For
Trial Riding"]
bill = 0

while True:
    print("\n1: View Bikes\n2: View Prices\n3: Place orders\n4: Exit")
    a = int(input("Choose an option: "))

    if a == 1:
        print("\nAvailable Bikes:")
        for bike in bikes:
            print(bike)

    elif a == 2:
        print("\nPrices:\n1. Hourly - Rs 100\n2. Daily - Rs 500\n3. 
Weekly - Rs 2500\nFamily pack - 30% discount on 3-5 bikes")

    elif a == 3:
        c = int(input("\nChoose rental type:\n1. Hourly\n2. Daily\n3. 
Weekly\n"))
        d = int(input("Enter number of bikes: "))

        if c == 1:
            bill += 100 * d
        elif c == 2:
            bill += 500 * d
        elif c == 3:
            bill += 2500 * d

        print("\nYour actual Bill is ", bill)

        if d in range(3, 6):
            dis = input("\nAvail family pack discount? (y/n): ")
            if dis.lower() == "y":
                bill *= 0.7
                print("\nThanks for purchasing. Your bill is ", bill)
                break

    elif a == 4:
        break

    else:
        print("\nInvalid option")
```

->PGM 9

```
fname = "File1.txt"
num_lines = 0
num_words = 0
num_chars = 0

with open(fname, 'r') as f:
    for line in f:
        words = line.split()
        num_lines += 1
        num_words += len(words)
        num_chars += len(line)

print("The total number of lines in the given file:", num_lines)
print("The total number of words in the given file:", num_words)
print("The total number of characters in the given file:", num_chars)
```

->PGM 10

```
import requests
from bs4 import BeautifulSoup
```

```
url = "https://en.wikipedia.org/wiki/Sachin_Tendulkar"
response = requests.get(url)
soup = BeautifulSoup(response.content, "html.parser")
images = soup.select("img")

for image in images:
    src = image.get("src")
    if src.startswith("//"):
        src = "https:" + src
    elif src.startswith("/"):
        src = "https://en.wikipedia.org" + src
    print(src)
```

->PGM 11

```
import requests
from bs4 import BeautifulSoup

url = "https://www.imdb.com/chart/top"
headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3"
}

response = requests.get(url, headers=headers)
soup = BeautifulSoup(response.content, "html.parser")
movies = soup.select(".lister-list .titleColumn a")

for movie in movies[:10]:
    link = "https://www.imdb.com" + movie.get("href")
    movie_response = requests.get(link, headers=headers)

    if movie_response.ok:
        movie_soup = BeautifulSoup(movie_response.content, "html.parser")

        try:
            movie_name = movie_soup.select_one(".title_wrapper
h1").get_text(strip=True).split(" (")[0]
        except AttributeError:
            movie_name = "N/A"

        try:
            movie_year = movie_soup.select_one(".title_wrapper h1
.title_year a").get_text(strip=True)
        except AttributeError:
            movie_year = "N/A"

        try:
            movie_summary =
movie_soup.select_one(".summary_text").get_text(strip=True)
        except AttributeError:
            movie_summary = "N/A"
        print(f"Name: {movie_name}")
        print(f"Year: {movie_year}")
        print(f"Summary: {movie_summary}")
        print("--------------------")
```

-> PGM 12

```
import turtle

def sierpinski(t, x, y, size, depth, change_depth):
    if depth == 0:
        t.penup()
        t.goto(x, y)
        t.pendown()
        for i in range(3):
            t.forward(size)
            t.left(120)
    else:
        sierpinski(t, x, y, size/2, depth-1, change_depth)
        sierpinski(t, x+size/2, y, size/2, depth-1, change_depth)
        sierpinski(t, x+size/4, y+(size/2)*(3**0.5)/2, size/2, depth-1,
change_depth)

        if depth == change_depth:
            t.fillcolor('magenta')
            t.begin_fill()
            sierpinski(t, x+size/4, y+(size/2)*(3**0.5)/2, size/2, 0,
change_depth)
            t.end_fill()

        t.fillcolor('red')
        t.begin_fill()
        sierpinski(t, x, y, size/2, 0, change_depth)
        t.end_fill()

        t.fillcolor('blue')
        t.begin_fill()
        sierpinski(t, x+size/2, y, size/2, 0, change_depth)
        t.end_fill()


t = turtle.Turtle()
t.speed(0)
change_depth = 2
sierpinski(t, -200, -200, 400, change_depth, change_depth)
turtle.done()
```

->PGM 13

```
import turtle

def koch_snowflake(t, x1, y1, x2, y2, depth):
    if depth == 0:
        t.penup()
        t.goto(x1, y1)
        t.pendown()
        t.goto(x2, y2)
    else:
        xa = x1 + (x2 - x1) / 3
        ya = y1 + (y2 - y1) / 3
        xb = x1 + 2 * (x2 - x1) / 3
        yb = y1 + 2 * (y2 - y1) / 3
        xc = (x1 + x2) / 2 - (y2 - y1) * (3**0.5) / 6
        yc = (y1 + y2) / 2 + (x2 - x1) * (3**0.5) / 6
        koch_snowflake(t, x1, y1, xa, ya, depth-1)
        koch_snowflake(t, xa, ya, xc, yc, depth-1)
        koch_snowflake(t, xc, yc, xb, yb, depth-1)
        koch_snowflake(t, xb, yb, x2, y2, depth-1)

t = turtle.Turtle()
t.speed(0)
depth = 2 # Change this value to specify the depth of recursion
size = 300
x1 = -size / 2
y1 = size * (3**0.5) / 6
x2 = size / 2
y2 = size * (3**0.5) / 6
x3 = 0
y3 = -size * (3**0.5) / 3
koch_snowflake(t, x1, y1, x2, y2, depth)
koch_snowflake(t, x2, y2, x3, y3, depth)
koch_snowflake(t, x3, y3, x1, y1, depth)
turtle.done()
```

->PGM 14

```
from mrjob.job import MRJob

class MovieSimilarities(MRJob):

    def mapper(self, _, line):
        try:
            twitter_id, movie_name, genre = line.split("::")
            yield genre, movie_name
        except ValueError:
            pass  # Handle lines with incorrect format

    def reducer(self, genre, movies):
        movie_list = list(movies)
        for i in range(len(movie_list)):
            for j in range(i + 1, len(movie_list)):
                similarity_score =
self.calculate_similarity(movie_list[i], movie_list[j])
                yield (movie_list[i], movie_list[j]), similarity_score

    def calculate_similarity(self, movie1, movie2):
        movie1 = movie1.lower()
        movie2 = movie2.lower()
        common_chars = set(movie1) & set(movie2)
        similarity_score = len(common_chars)
        return similarity_score

if __name__ == '__main__':
    MovieSimilarities.run()


-> PGM 15

import tkinter as tk

def calculate_bmi():
    try:
        weight = float(weight_entry.get())
        height = float(height_entry.get())
        if height == 0:  # Check for zero height to avoid division by
zero
            bmi_label.config(text="Invalid Height")
            return

        bmi = round(weight / (height ** 2), 2)
        bmi_label.config(text=f"BMI: {bmi}")

    except ValueError: #Handles non-numeric input
        bmi_label.config(text="Invalid Input")



root = tk.Tk()
root.title("BMI Calculator")

weight_label = tk.Label(root, text="Weight (kg):")
weight_label.grid(row=0, column=0)

weight_entry = tk.Entry(root)
weight_entry.grid(row=0, column=1)

height_label = tk.Label(root, text="Height (m):")
height_label.grid(row=1, column=0)

height_entry = tk.Entry(root)
height_entry.grid(row=1, column=1)

calculate_button = tk.Button(root, text="Calculate BMI",
command=calculate_bmi)
calculate_button.grid(row=2, column=0, columnspan=2)

bmi_label = tk.Label(root, text="BMI:")
bmi_label.grid(row=3, column=0, columnspan=2)

root.mainloop()
```