```python
# 7 Write a Python program to generate a random alphabetical character,
alphabetical string
# and alphabetical string of a fixed length. Use random.choice()


import random
import string

rndchoice = random.choice(string.ascii_letters)
print("Random alphabetical character:", rndchoice)

random_length = random.randint(5, 15)
random_string = ''
for _ in range(random_length):
    random_string += random.choice(string.ascii_letters)
print("Random alphabetical string:", random_string)

fixed_length = 10
fixed_length_string = ''
for _ in range(fixed_length):
    fixed_length_string += random.choice(string.ascii_letters)
print("Random alphabetical string of fixed length:", fixed_length_string)
```

```python
# 8 Create a child class Bus that will inherit all of the variables and
methods of the Vehicle class
class Vehicle:
    def __init__(self, color, max_speed):
        self.color = color
        self.max_speed = max_speed

    def display_info(self):
        print(f"Color: {self.color}, Max Speed: {self.max_speed}")


class Bus(Vehicle):
    def __init__(self, color, max_speed, capacity):
        # Call the constructor of the parent class (Vehicle) to initialize
color and max_speed
        super().__init__(color, max_speed)
        self.capacity = capacity

    def display_info(self):
        # Call the display_info method of the parent class (Vehicle)
        super().display_info()
        print(f"Capacity: {self.capacity}")
```

```python
# Example usage
bus = Bus("Yellow", 60, 50)
bus.display_info()
```

```python
# 9 Create a Bus class that inherits from the Vehicle class. Give the capacity argument
# of Bus.seating_capacity() a default value of 50.
class Vehicle:
    def __init__(self, color, max_speed):
        self.color = color
        self.max_speed = max_speed

    def display_info(self):
        print(f"Color: {self.color}, Max Speed: {self.max_speed}")


class Bus(Vehicle):
    def __init__(self, color, max_speed, capacity=50):
        super().__init__(color, max_speed)
        self.capacity = capacity

    def seating_capacity(self):
        print(f"Seating Capacity of the Bus: {self.capacity}")


# Example usage
bus = Bus("Yellow", 60)
bus.display_info()
bus.seating_capacity()
```

```python
# 10 Create a Bus child class that inherits from the Vehicle class. The default fare charge of any
# vehicle is seating capacity * 100. If Vehicle is Bus instance, we need to add an extra 10% on
# full fare as a maintenance charge. So total fare for bus instance will become the final amount =
# total fare + 10% of the total fare.
class Vehicle:
    def __init__(self, color, max_speed, capacity):
        self.color = color
        self.max_speed = max_speed
        self.capacity = capacity
```

```
    def fare_charge(self):
        fare = self.capacity * 100
        return fare


class Bus(Vehicle):
    def __init__(self, color, max_speed, capacity):
        super().__init__(color, max_speed, capacity)

    def fare_charge(self):
        fare = super().fare_charge()
        if isinstance(self, Bus):
            fare += fare * 0.1  # Adding 10% maintenance charge for buses
        return fare


# Example usage
bus = Bus("Yellow", 60, 50)
print("Total fare for bus:", bus.fare_charge())
```

```
# 11 Write a Python class named Student with two attributes student_name,
marks. Modify the
# attribute values of the said class and print the original and modified
values of the said
# attributes.
class Student:
    def __init__(self, student_name, marks):
        self.student_name = student_name
        self.marks = marks

# Create an instance of the Student class
student = Student("John", 85)

# Print the original attribute values
print("Original Student Name:", student.student_name)
print("Original Marks:", student.marks)

# Modify the attribute values
student.student_name = "Alice"
student.marks = 90

# Print the modified attribute values
print("Modified Student Name:", student.student_name)
print("Modified Marks:", student.marks)
```

```python
# 12 Write a Python program to match a string that contains only upper and
lowercase
# letters, numbers, and underscores.
import re

def match_string(text):
    pattern = r'^[a-zA-Z0-9_]*$'
    if re.match(pattern, text):
        return True
    else:
        return False

# Test cases
strings_to_test = ["Hello_World123", "abcDEF456", "123_456", "special@chars",
"Spaces Not Allowed"]
for text in strings_to_test:
    if match_string(text):
        print(f"'{text}' matches the pattern.")
    else:
        print(f"'{text}' does not match the pattern.")
```

```python
# 13] Write a python program to validate the password by using regular
expression.
# a. Complexity requirement is that we need at least one capital letter, one
number and one
# special character.
# b. We also need the length of the password to be between 8 and 18.
import re

def validate_password(password):
    # Ensure length is between 8 and 18 characters
    if len(password) < 8 or len(password) > 18:
        return False

    # Ensure at least one capital letter, one number, and one special
character
    pattern = r'^(?=.*[A-Z])(?=.*\d)(?=.*[!@#$%^&*()_+{}|:<>?]).+$'
    if re.match(pattern, password):
        return True
    else:
        return False

# Test cases
passwords_to_test = ["Abc123!@#", "Password123", "short",
"Toolongpassword123456", "NoSpecialCharacter123"]
for password in passwords_to_test:
    if validate_password(password):
```

```
        print(f"'{password}' is a valid password.")
    else:
        print(f"'{password}' is not a valid password.")
```

```
# 14 Write a python program to validate the URL by using regular expression.
16]
import re

def validate_url(url):
    # Regular expression pattern for URL validation
    pattern = r'^(https?|ftp)://[^\s/$.?#].[^\s]*$'
    if re.match(pattern, url):
        return True
    else:
        return False

# Test cases
urls_to_test = [
    "http://www.example.com",
    "https://example.com/page",
    "ftp://ftp.example.com/file",
    "invalid-url",
    "http://www.invalid url.com",
    "https://www.invalid?url.com"
]

for url in urls_to_test:
    if validate_url(url):
        print(f"'{url}' is a valid URL.")
    else:
        print(f"'{url}' is not a valid URL.")
```

```
# 15 Write a python program to validate an email address by using regular
expression.
import re

def validate_email(email):
    # Regular expression pattern for email validation
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    if re.match(pattern, email):
        return True
    else:
        return False
```

```python
# Test cases
emails_to_test = [
    "example@example.com",
    "user123@example.co.uk",
    "user.name@example.org",
    "invalid-email",
    "invalid@.com",
    "@example.com",
    "user@examplecom"
]

for email in emails_to_test:
    if validate_email(email):
        print(f"'{email}' is a valid email address.")
    else:
        print(f"'{email}' is not a valid email address.")
```

```python
# 16 Write a python program which consists of - try, except, else, finally blocks.
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("Error: Division by zero!")
    else:
        print("Division successful.")
        print("Result:", result)
    finally:
        print("Executing 'finally' block.")


# Test cases
print("Test Case 1:")
divide(10, 2)

print("\nTest Case 2:")
divide(10, 0)
```

```python
# 17 Write a python program which raises the exception with a message.
def divide(x, y):
    if y == 0:
        raise ZeroDivisionError("Error: Division by zero is not allowed!")
```

```
    else:
        return x / y

# Test cases
try:
    result = divide(10, 0)
except ZeroDivisionError as e:
    print(e)
else:
    print("Result:", result)
```

```
# 18 Write a Python multithreading program to print the thread name and
corresponding process
# for each task (assume that there are four tasks).
import threading
import os

def task(task_number):
    print(f"Task {task_number} is running in thread:
{threading.current_thread().name} (Process ID: {os.getpid()})")

if __name__ == "__main__":
    # Create four threads for four tasks
    threads = []
    for i in range(4):
        thread = threading.Thread(target=task, args=(i+1,))
        threads.append(thread)
        thread.start()

    # Wait for all threads to finish
    for thread in threads:
        thread.join()

    print("All tasks are completed.")
```

```
# 19 Write a Python multithreading program which creates two threads, one for
calculating the
# square of a given number and other for calculating the cube of a given
number.
import threading

def calculate_square(number):
```

```
    print(f"Square of {number}: {number ** 2}")

def calculate_cube(number):
    print(f"Cube of {number}: {number ** 3}")

if __name__ == "__main__":
    number = 5

    # Create two threads, one for square and one for cube
    square_thread = threading.Thread(target=calculate_square, args=(number,))
    cube_thread = threading.Thread(target=calculate_cube, args=(number,))

    # Start both threads
    square_thread.start()
    cube_thread.start()

    # Wait for both threads to finish
    square_thread.join()
    cube_thread.join()

    print("Main thread finished.")
```

```
# 20 Given a file called myfile.txt which contains the text: “Python is object
oriented programming
# language”. Write a program in Python that transforms the content of the file
by writing each word
# in a separate line.
# Open the input file in read mode
with open("myfile.txt", "r") as file:
    # Read the content of the file
    content = file.read()
    # Split the content into words
    words = content.split()

# Open a new file in write mode
with open("transformed_file.txt", "w") as new_file:
    # Write each word on a separate line
    for word in words:
        new_file.write(word + "\n")

print("Transformation complete. Check transformed_file.txt for the result.")
```

```python
# 21] Write a Python program that displays the longest word found in a text file
def longest_word(filename):
    with open(filename, 'r') as file:
        # Read the content of the file
        content = file.read()
        # Split the content into words
        words = content.split()
        # Find the longest word
        longest = max(words, key=len)
    return longest

# Test the function with a file named "example.txt"
filename = "example.txt"  # Replace "example.txt" with the name of your text file
longest = longest_word(filename)
print("Longest word in the file:", longest)
```

```python
# 22 Write a function in python that allows you to count the frequency of repetition of each word
# found in a given file.
def count_word_frequency(filename):
    word_frequency = {}
    with open(filename, 'r') as file:
        # Read the content of the file
        content = file.read()
        # Split the content into words
        words = content.split()
        # Count the frequency of each word
        for word in words:
            if word in word_frequency:
                word_frequency[word] += 1
            else:
                word_frequency[word] = 1
    return word_frequency

# Test the function with a file named "example.txt"
filename = "example.txt"  # Replace "example.txt" with the name of your text file
word_frequency = count_word_frequency(filename)

# Print the frequency of each word
for word, frequency in word_frequency.items():
    print(f"'{word}': {frequency}")
```

```python
# 23 Write a Python program which allows you to extract the content of a file from the 3rd line to
# the 7th line and save it in another file called extract_content.txt.
def extract_content(input_file, output_file):
    with open(input_file, 'r') as file:
        # Read all lines from the input file
        lines = file.readlines()

    # Extract lines from 3rd to 7th
    extracted_lines = lines[2:7]

    # Write extracted lines to the output file
    with open(output_file, 'w') as file:
        file.writelines(extracted_lines)

# Test the function with a file named "input_file.txt"
input_file = "input_file.txt"  # Replace "input_file.txt" with the name of your input file
output_file = "extract_content.txt"
extract_content(input_file, output_file)

print(f"Content from {input_file} extracted and saved in {output_file}.")
```

```python
# 24 Create the following DataFrame Sales containing year wise sales figures for five
# salespersons in INR. Use the years as column labels, and salesperson names as row labels.
# 2018 2019 2020 2021
# Kapil 110 205 177 189 Kamini 130 165 175 190 Shikhar 115 206 157 179 Mohini 118 198
# 183 169
# 1. Create the DataFrame.
# 2. Display the row labels of Sales.
# 3. Display the column labels of Sales.
# 4. Display the data types of each column of Sales.
# 5. Display the dimensions, shape, size and values of Sales
import pandas as pd

# Create the DataFrame Sales
data = {
    '2018': [110, 130, 115, 118],
    '2019': [205, 165, 206, 198],
    '2020': [177, 175, 157, 183],
    '2021': [189, 190, 179, 169]
```

```
}
salespersons = ['Kapil', 'Kamini', 'Shikhar', 'Mohini']
sales = pd.DataFrame(data, index=salespersons)

# Display the row labels of Sales
print("Row labels of Sales:")
print(sales.index)

# Display the column labels of Sales
print("\nColumn labels of Sales:")
print(sales.columns)

# Display the data types of each column of Sales
print("\nData types of each column of Sales:")
print(sales.dtypes)

# Display the dimensions, shape, size, and values of Sales
print("\nDimensions of Sales:")
print(sales.ndim)
print("\nShape of Sales:")
print(sales.shape)
print("\nSize of Sales:")
print(sales.size)
print("\nValues of Sales:")
print(sales.values)
```

```
# 25 Plot the following data on a line chart and customize the chart according
to the belowgiven instructions:
# Month January February March April May Sales 510 350 475 580 600 Weekly
Sales Report
# 1. Write a title for the chart “The Monthly Sales Report“
# 2. Write the appropriate titles of both the axes
# 3. Write code to Display legends
# 4. Display blue color for the line
# 5. Use the line style – dashed
# 6. Display diamond style markers on data points
import matplotlib.pyplot as plt

# Data
months = ['January', 'February', 'March', 'April', 'May']
sales = [510, 350, 475, 580, 600]

# Plot the data
plt.plot(months, sales, color='blue', linestyle='--', marker='D',
markersize=8, label='Sales')
```

```
# Customize the chart
plt.title('The Monthly Sales Report')
plt.xlabel('Month')
plt.ylabel('Sales')
plt.legend()

# Display the chart
plt.show()
```

```
#  26 Observe following data and plot data according to given instructions:
# Batsman 2017 2018 2019 2020 Virat Kohli 2501 1855 2203 1223 Steve Smith 2340
2250 2003
# 1153 Babar Azam 1750 2147 1896 1008 Rohit Sharma 1463 1985 1854 1638 Kane
Williamson
# 1256 1785 1874 1974 Jos Butler 1125 1853 1769 1436
# 1. Create a bar chart to display data of Virat Kohli & Rohit Sharma.
# 2. Customize the chart in this manner
# 1. Use different widths
# 2. Use different colors to represent different years score
# 3. Display appropriate titles for axis and chart
# 4. Show legends
# 5. Create a bar chart to display data of Steve Smith, Kane Williamson & Jos
Butler.
# Customize Chart as per your wish.
# 6. Display data of all players for the specific year.
import matplotlib.pyplot as plt

# Data
batsmen = ['Virat Kohli', 'Steve Smith', 'Babar Azam', 'Rohit Sharma', 'Kane
Williamson', 'Jos Butler']
years = ['2017', '2018', '2019', '2020']
scores = {
    'Virat Kohli': [2501, 1855, 2203, 1223],
    'Steve Smith': [2340, 2250, 2003, 1153],
    'Babar Azam': [1750, 2147, 1896, 1008],
    'Rohit Sharma': [1463, 1985, 1854, 1638],
    'Kane Williamson': [1256, 1785, 1874, 1974],
    'Jos Butler': [1125, 1853, 1769, 1436]
}

# Plot for Virat Kohli & Rohit Sharma
plt.figure(figsize=(10, 6))  # Set the figure size

for i, (batsman, color) in enumerate(zip(['Virat Kohli', 'Rohit Sharma'],
['blue', 'orange'])):
```

```
    plt.bar([x + i * 0.2 for x in range(len(years))], scores[batsman],
width=0.2, color=color, label=batsman)

plt.title('Virat Kohli & Rohit Sharma - Yearly Scores')
plt.xlabel('Year')
plt.ylabel('Score')
plt.xticks(range(len(years)), years)
plt.legend()
plt.show()

# Plot for Steve Smith, Kane Williamson & Jos Butler
plt.figure(figsize=(10, 6))  # Set the figure size

for i, batsman in enumerate(['Steve Smith', 'Kane Williamson', 'Jos Butler']):
    plt.bar([x + i * 0.2 for x in range(len(years))], scores[batsman],
width=0.2, label=batsman)

plt.title('Steve Smith, Kane Williamson & Jos Butler - Yearly Scores')
plt.xlabel('Year')
plt.ylabel('Score')
plt.xticks(range(len(years)), years)
plt.legend()
plt.show()

# Display data of all players for the specific year (e.g., 2019)
year = '2019'
plt.figure(figsize=(10, 6))  # Set the figure size

for batsman in batsmen:
    plt.bar(batsman, scores[batsman][years.index(year)], label=batsman)

plt.title(f'Yearly Scores of All Players in {year}')
plt.xlabel('Batsmen')
plt.ylabel('Score')
plt.legend()
plt.xticks(rotation=45)
plt.show()
```

```
# 27] WAP to create a 3*3 numpy array with all the elements as per the user
choice and print
# the sum of all elements of the array.
import numpy as np

# Function to create a 3x3 array with user input elements
def create_array():
    elements = []
    for _ in range(3):
```

```
        row = []
        for _ in range(3):
            element = float(input("Enter element for the array: "))
            row.append(element)
        elements.append(row)
    return np.array(elements)

# Create the array
array = create_array()

# Print the array
print("Array:")
print(array)

# Print the sum of all elements in the array
print("Sum of all elements:", np.sum(array))
```

```
# 28] WAP to perform basic arithmetic operations on 1D and 2D array .
import numpy as np

# Create 1D array
array1d = np.array([1, 2, 3, 4, 5])

# Create 2D array
array2d = np.array([[1, 2, 3], [4, 5, 6]])

# Addition
print("Addition:")
print("1D array + 5:", array1d + 5)
print("2D array + 2:")
print(array2d + 2)

# Subtraction
print("\nSubtraction:")
print("1D array - 2:", array1d - 2)
print("2D array - 1:")
print(array2d - 1)

# Multiplication
print("\nMultiplication:")
print("1D array * 3:", array1d * 3)
print("2D array * 2:")
print(array2d * 2)

# Division
```

```
print("\nDivision:")
print("1D array / 2:", array1d / 2)
print("2D array / 3:")
print(array2d / 3)

# Element-wise square root
print("\nSquare Root:")
print("Square root of 1D array:", np.sqrt(array1d))
print("Square root of 2D array:")
print(np.sqrt(array2d))

# Element-wise exponential
print("\nExponential:")
print("Exponential of 1D array:", np.exp(array1d))
print("Exponential of 2D array:")
print(np.exp(array2d))
```

```
# 29]Write a Menu Driver Program to add, display, update, delete and exit in a
student database
# containing Student_id,Student_name,Course through Python-MongoDB
connectivity.
import pymongo

# Function to connect to MongoDB
def connect_to_mongodb():
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    db = client["student_database"]
    collection = db["students"]
    return collection

# Function to add a student record
def add_student(collection):
    student_id = input("Enter Student ID: ")
    student_name = input("Enter Student Name: ")
    course = input("Enter Course: ")
    student = {"Student_id": student_id, "Student_name": student_name,
"Course": course}
    collection.insert_one(student)
    print("Student added successfully!")

# Function to display all student records
def display_students(collection):
    students = collection.find()
    for student in students:
        print(student)
```

```python
# Function to update a student record
def update_student(collection):
    student_id = input("Enter Student ID to update: ")
    new_course = input("Enter new Course: ")
    collection.update_one({"Student_id": student_id}, {"$set": {"Course":
new_course}})
    print("Student record updated successfully!")

# Function to delete a student record
def delete_student(collection):
    student_id = input("Enter Student ID to delete: ")
    collection.delete_one({"Student_id": student_id})
    print("Student record deleted successfully!")

# Main function
def main():
    collection = connect_to_mongodb()
    while True:
        print("\nMenu:")
        print("1. Add Student")
        print("2. Display Students")
        print("3. Update Student")
        print("4. Delete Student")
        print("5. Exit")
        choice = input("Enter your choice: ")
        if choice == "1":
            add_student(collection)
        elif choice == "2":
            display_students(collection)
        elif choice == "3":
            update_student(collection)
        elif choice == "4":
            delete_student(collection)
        elif choice == "5":
            print("Exiting the program...")
            break
        else:
            print("Invalid choice! Please enter a valid option.")

if __name__ == "__main__":
    main()
```

```python
# 30] Demonstrate step by step MongoDB connection in Python
import pymongo
```

```python
# Step 1: Establish a connection to MongoDB
def Conn():
    try:
        # Connect to MongoDB server (default host and port)
        client = pymongo.MongoClient("mongodb://localhost:27017/")
        print("Connected to MongoDB successfully!")
        return client
    except pymongo.errors.ConnectionFailure as e:
        print("Could not connect to MongoDB:", e)

# Step 2: Connect to a specific database
def Conn_to_database(client, database_name):
    try:
        # Access the specified database
        db = client[database_name]
        print(f"Connected to database '{database_name}' successfully!")
        return db
    except Exception as e:
        print("Error connecting to database:", e)

# Step 3: Access a specific collection within the database
def access_collection(db, collection_name):
    try:
        # Access the specified collection within the database
        collection = db[collection_name]
        print(f"Accessed collection '{collection_name}' successfully!")
        return collection
    except Exception as e:
        print("Error accessing collection:", e)

# Step 4: Perform CRUD operations or other tasks with the collection
def main():
    # Step 1: Connect to MongoDB
    client = Conn()

    # Step 2: Connect to a specific database
    database_name = "my_database"
    db = Conn_to_database(client, database_name)

    # Step 3: Access a specific collection within the database
    collection_name = "my_collection"
    collection = access_collection(db, collection_name)

    # Step 4: Perform CRUD operations or other tasks with the collection
    # For example, you can insert documents into the collection, query
documents, update documents, delete documents, etc.
```

```
    # Close the MongoDB connection when done
    client.close()
    print("Connection to MongoDB closed.")

if __name__ == "__main__":
    main()
```

```
# 31] Write a Menu Driver Program to add, display, search, sort and exit in
book database
# containing
# Book_id, Book_name, Book_author through Python-MongoDB connectivity.
import pymongo

# Function to connect to MongoDB
def Conn():
    client = pymongo.MongoClient("mongodb://localhost:27017/")
    db = client["book_database"]
    collection = db["books"]
    return collection

# Function to add a book record
def add_book(collection):
    book_id = input("Enter Book ID: ")
    book_name = input("Enter Book Name: ")
    book_author = input("Enter Book Author: ")
    book = {"Book_id": book_id, "Book_name": book_name, "Book_author":
book_author}
    collection.insert_one(book)
    print("Book added successfully!")

# Function to display all book records
def display_books(collection):
    books = collection.find()
    for book in books:
        print(book)

# Function to search for a book by name
def search_book(collection):
    book_name = input("Enter Book Name to search: ")
    books = collection.find({"Book_name": book_name})
    for book in books:
        print(book)

# Function to sort books by name
def sort_books(collection):
```

```
    books = collection.find().sort("Book_name")
    for book in books:
        print(book)

# Main function
def main():
    collection = Conn()
    while True:
        print("\nMenu:")
        print("1. Add Book")
        print("2. Display Books")
        print("3. Search Book")
        print("4. Sort Books")
        print("5. Exit")
        choice = input("Enter your choice: ")
        if choice == "1":
            add_book(collection)
        elif choice == "2":
            display_books(collection)
        elif choice == "3":
            search_book(collection)
        elif choice == "4":
            sort_books(collection)
        elif choice == "5":
            print("Exiting the program...")
            break
        else:
            print("Invalid choice! Please enter a valid option.")

if __name__ == "__main__":
    main()
```