****



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**B. E. (COMPUTER SCIENCE AND ENGINEERING)**

**IV Semester**



**Name** : ...................................................................................................................

**Reg. No.** : ....................................................................................................................





## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**B. E. (COMPUTER SCIENCE AND ENGINEERING)**

**IV Semester**



Certified that this is a bona-fide record of work done by Mr./Ms............................................................................................

Reg. No. .......................................... of B.E.(Computer Science and Engineering) in the 22CSCP410 – Python Programming Lab during the even semester of the academic year 2024–25.

Staff-in-charge Internal Examiner

Place: Annamalainagar

Date :

External Examiner

| **CONTENTS** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Date** | **List of Experiments** | **Page No** | **Mark** | **Sign** |
| 01 |  | Tuples | 1 |  |  |
| 02 |  | List | 2 |  |  |
| 03 |  | Set | 4 |  |  |
| 04 |  | Dictionary | 5 |  |  |
| 05 |  | Conditional Statement | 6 |  |  |
| 06 |  | String Manipulation | 8 |  |  |
| 07 |  | Extracting Titles using Lamda Function | 9 |  |  |
| 08 |  | Student Grade Calculator with Class and Objects | 11 |  |  |
| 09 |  | Library Management with Class and Objects | 13 |  |  |
| 10 |  | Simple Bank Account using Operator Overloading | 16 |  |  |
| 11 |  | Inheritance Vehicle Management System | 18 |  |  |
| 12 |  | File Handling on Log Files | 21 |  |  |
| 13 |  | File Handling on CSV Files | 24 |  |  |
| 14 |  | Calculator using Exception Handling | 27 |  |  |
| 15 |  | Numerical Data Processing using Pandas | 30 |  |  |
| 16 |  | E-Commerce Sales Analysis with Matplotlib | 32 |  |  |
| 17 |  | Tic-Tac-Toe Game with Tkinter | 39 |  |  |
| **Total:** | | | |  | |
| **Average:** | | | |  | |

# Annamalai University

## Department of Computer Science and Engineering

**VISION**

To provide a congenial ambience for individuals to develop and blossom as academically superior, socially conscious and nationally responsible citizens.

**MISSION**

* Impart high quality computer knowledge to the students through a dynamic scholastic environment wherein they learn to develop technical, communication and leadership skills to bloom as a versatile professional.
* Develop life-long learning ability that allows them to be adaptive and responsive to the changes in career, society, technology, and environment.
* Build student community with high ethical standards to undertake innovative research and development in thrust areas of national and international needs.
* Expose the students to the emerging technological advancements for meeting the demands of the industry.

**PROGRAM EDUCATIONAL OBJECTIVES (PEOs)**

| PEO | PEO Statements |
| --- | --- |
| PEO1 | To prepare the graduates with the potential to get employed in the right role and/or become  entrepreneurs to contribute to the society. |
| PEO2 | To provide the graduates with the requisite knowledge to pursue higher education and carry  out research in the field of Computer Science. |
| PEO3 | To equip the graduates with the skills required to stay motivated and adapt to the dynamically  changing world so as to remain successful in their career. |
| PEO4 | To train the graduates to communicate effectively, work collaboratively and exhibit high  levels of professionalism and ethical responsibility. |

**PROGRAM OUTCOMES (POs)**

| **S. No.** | Program Outcomes |
| --- | --- |
| **PO1** | **Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering  fundamentals, and an engineering specialization to the solution of complex engineering problems. |
| **PO2** | **Problem Analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles  of mathematics, natural sciences and engineering sciences. |
| **PO3** | **Design/Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural,  societal, and environmental considerations. |
| **PO4** | **Conduct Investigations of Complex Problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data,  and synthesis of the information to provide valid conclusions. |
| **PO5** | **Modern Tool Usage:** Create, select, and apply appropriate techniques, resources, and  modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations. |
| **PO6** | **The Engineer and Society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent  responsibilities relevant to the professional engineering practice. |
| **PO7** | **Environment and Sustainability:** Understand the impact of the professional  engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development. |
| **PO8** | **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities  and norms of the engineering practice. |
| **PO9** | **Individual and Team Work:** Function effectively as an individual, and as a member  or leader in diverse teams, and in multidisciplinary settings. |
| **PO10** | **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and  give and receive clear instructions. |

| **PO11** | **Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary  environments. |
| --- | --- |
| **PO12** | **Life-long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological  change. |

**PROGRAM SPECIFIC OUTCOMES (PSOs)**

| **S.no** | **Program Specific Outcomes** |
| --- | --- |
| **PSO1** | Acquire the ability to understand basic sciences, humanity sciences, basic engineering sciences and fundamental core courses in Computer Science and Engineering to realize and appreciate real life problems in diverse fields for  proficient design of computer based systems of varying complexity. |
| **PSO2** | Learn specialized courses in Computer Science and Engineering to build up the  aptitude for applying typical practices and approaches to deliver quality products intended for business and industry requirements. |
| **PSO3** | Apply technical and programming skills in Computer Science and Engineering essential for employing current techniques in software development crucial in industries, to create pioneering career paths for pursuing higher studies, research  and to be an entrepreneur. |

\*\*\*

# Rubrics for Laboratory Examination (Internal/External)

**(Internal:** Two tests - 15 marks each, **External:** Two questions - 25 marks each)

| **Rubric** | **Poor Up to (1/2)** | **Average Up to (2/4)** | **Good Up to (3/6)** | **Excellent Up to (5/8\*)** |
| --- | --- | --- | --- | --- |
| **Syntax and Logic Ability to understand, specify the data structures appropriate for the problem**  **domain** | Program does not compile with typographical errors and incorrect logic leading to infinite loops. | Program compiles that signals major syntactic errors and logic shows severe errors. | Program compiles with minor syntactic errors and logic is mostly correct with occasional errors. | Program compiles with evidence of good syntactic understanding of the syntax and logic used. |
| **Modularity Ability to decompose a problem into coherent and reusable functions, files, classes, or objects (as appropriate for the programming language**  **and platform).** | Program is one big Function or is decomposed in ways that make little/no sense. | Program is decomposed into units of appropriate size, but they lack coherence or reusability.  Program contains unnecessary repetition. | Program is decomposed  into coherent units, but may still contain some unnecessary repetition. | Program is decomposed  into coherent and reusable units, and unnecessary repetition are eliminated. |
| **Clarity and Completeness Ability to code formulae and algorithms that produce appropriate results. Ability to apply rigorous test case analysis to the problem domain.** | Program does not produce appropriate results for most inputs.  Program shows little/no ability to apply different test cases. | Program approaches appropriate results for  most inputs, but contain some miscalculations. Program shows evidence of test case analysis, but missing significant test cases or mistaken some  test cases. | Program produces appropriate results for most inputs.  Program shows evidence of test case analysis that is mostly complete, but missed to handle all possible test cases. | Program produces appropriate results for all inputs tested.  Program shows evidence  of excellent test case analysis, and all possible cases are handled appropriately. |

\* 8 marks for syntax and logic, 8 marks for modularity, and 9 marks for Clarity and Completeness.

**Rubric for CO3**

| **Rubric for CO3 in Laboratory Courses** | | | | |
| --- | --- | --- | --- | --- |
| **Rubric** | **Distribution of 10 Marks for CIE/SEE Evaluation Out of 40/60 Marks** | | | |
| **Up To 2.5 Marks** | **Up To 5 Marks** | **Up To 7.5 Marks** | **Up To 10 marks** |
| **Demonstrate** | Poor listening and | Showed better | Demonstrated | Demonstrated |
| **an ability to** | communication | communication | good | excellent |
| **listen and** | skills. Failed to | skill by relating | communication | communication |
| **answer the** | relate the | the problem with | skills by relating | skills by relating |
| **viva** | programming | the programming | the problem with | the problem with |
| **questions** | skills needed for | skills acquired | the programming | the programming |
| **related to** | solving the | but the | skills acquired | skills acquired and |
| **programming** | problem. | description | with few errors. | have been |
| **skills needed** |  | showed serious |  | successful in |
| **for solving** |  | errors. |  | tailoring the |
| **real-world** |  |  |  | description. |
| **problems in** |  |  |  |  |
| **Computer** |  |  |  |  |
| **Science and** |  |  |  |  |
| **Engineering.** |  |  |  |  |

**Ex No: 01 TUPLES**

**Date: 24/01/2024**

**Aim:**

To create a python function that takes the list and returns a new dictionary where the keys are student names and the values are their average scores using tuples unpacking and list comprehension.

**Algorithm:**

1. In this code, we use list comprehension to iterate through each tuple in list of students.
2. Using tuples unpacking, we assign the first element of tuple to variable ‘name’ and the remaining elements to variable ‘score’.

3) Then, we calculate the average score by summing up the scores and dividing by number of scores.

4) We create a dictionary comprehension to build the dictionary with student names as keys and their scores as values.

5) You can call the ‘average\_scores’ function with your list of tuples and it will return the desired dictionary.

**Source code:**

def average\_scores(studentd):

scr\_dict = {}

for name, \*scr in studentd:

avgscr = sum(scr) / len(scr)

scr\_dict[name] = avgscr

return scr\_dict

std1 = [('Abi', 85, 90, 92), ('Bala', 78, 89, 90), ('Dharshini', 92, 88, 95)]

c = average\_scores(std1)

print(c)

**Sample Input and Output:**

{‘Abi’:89.0,‘Bala’:85.6666667,‘Dharshini’:91.6666667}

**Result:**

Thus, a python function that converts a list of tuples into dictionary is successfully created.

**Ex No: 02 LIST**

**Date: 24/01/2024**

**Aim:**

To create a python function that takes a list of integers as input and returns a list all unique combinations of two numbers that sum to a prime number.

**Algorithm:**

1) Define a function get\_prime\_sum\_combinations(numbers) that takes a list of integers as input.

2) Define a helper function is\_prime(n) that checks if a number n is prime.

- If n is less than 2, return False.

- Iterate from 2 to the square root of n and check if n is divisible by any number in that range. If it is, return False.

- If no divisor is found, return True.

3) Initialize an empty list called combinations to store the unique combinations of two numbers.

4) Iterate over the range of the length of the numbers list, using i as the index of the first number.

- Within this loop, iterate over the range from i+1 to the length of the numbers list, using j as the index of the second number.

- Create a tuple called pair with the two numbers at indices i and j.

- Check if the sum of the pair is a prime number using the is\_prime() function.

- If it is prime, append the pair to the combinations list.

5) Return the combinations list.

6) Test the function by calling get\_prime\_sum\_combinations() with a list of integers.

**Source code:**

def get\_prime\_sum\_combinations(numbers):

def is\_prime(n):

if n < 2:

return False

for i in range(2, int(n\*\*0.5) + 1):

if n % i == 0:

return False

return True # Return True if the number is prime

combinations = []

for i in range(len(numbers)):

for j in range(i+1, len(numbers)):

pair = (numbers[i], numbers[j])

if is\_prime(sum(pair)):

combinations.append(pair)

return combinations

numbers = [76, 87, 98, 78, 66, 87, 45, 23]

print(get\_prime\_sum\_combinations(numbers))

**Sample Input and Output:**

[(76,87), (76,87), (78,23), (66,23)]

**Results:**

Thus, a python function that converts a list of integers into a list of all unique combinations of two numbers that sum to a prime number is successfully created.

**Ex No: 03 SET**

**Date: 31/01/2024**

**Aim:**

To create a python function that takes two sets as input and returns a new set containing elements that are common to both sets.

**Algorithm:**

1) Start by defining the find\_common\_elements function that takes two sets, set1 and set2, as input.

2) Inside the function, use the & operator to find the intersection of set1 and set2.

3) Return the result of the intersection operation.

4) Outside the function, create two sets, set\_a and set\_b, with the desired elements.

5) Call the find\_common\_elements function, passing set\_a and set\_b as arguments.

6) Store the result in a variable, such as result.

7) Print the value of result to display the common elements between the two sets.

**Source code:**

def find\_common\_elements(set1, set2):

return set1 & set2

set\_a = {1, 2, 3, 4, 5}

set\_b = {3, 4, 5, 6, 7}

result = find\_common\_elements(set\_a, set\_b)

print(result)

**Sample Input and Output:**

{3,4,5}

**Results:**

Thus, a python function that takes two sets as input and returns a new set containing common elements is successfully created.

**Ex No: 04 DICTIONARY**

**Date: 31/01/2024**

**Aim:**

To create a python function that takes two dictionaries as input and returns a new dictionary containing merged key-value pairs. If there are common keys, sum the corresponding values.

**Algorithm:**

1) Start by defining the merge\_dictionaries function that takes two dictionaries, dict1 and dict2, as input.

2) Inside the function, use a dictionary comprehension to create a new dictionary.

3) Iterate over the union of keys from dict1 and dict2 using the set function and the | operator.

4) For each key, use the get method to retrieve the corresponding values from dict1 and dict2. If a key is missing in either dictionary, default to 0.

5) Add the values together and assign the result as the value for the key in the new dictionary.

6) Return the new dictionary.

7) Outside the function, create two dictionaries, dict\_a and dict\_b, with the desired key-value pairs.

8) Call the merge\_dictionaries function, passing dict\_a and dict\_b as arguments.

9) Store the result in a variable, such as result.

10) Print the value of result to display the merged dictionary.

**Source code:**

def merge\_dictionaries(dict1, dict2):

return {key: dict1.get(key, 0) + dict2.get(key, 0) for key in set(dict1) | set(dict2)}

dict\_a = {'a': 1, 'b': 2, 'c': 3}

dict\_b = {'b': 3, 'c': 4, 'd': 5}

result = merge\_dictionaries(dict\_a, dict\_b)

print(result)

**Sample Input and Output:**

{‘a’:1,’b’:5,’c’:7,’d’:5}

**Results:**

Thus, a python function of merged new dictionary is successfully created.

**Ex No: 05 CONDITIONAL STATEMENT**

**Date: 07/02/2024**

**Aim:**

To write a Python program that calculates a student’s grade using if statements.

**Algorithm:**

1. Define the weights for exams, assignments, and participation.

2. Input the scores for exams, assignments, and participation.

3. Calculate the overall score using the weighted averages.

4. Apply conditions to determine if any individual component score is below 40 or if participation score is 0.

5. Determine the final grade based on the overall score and the specified grade ranges.

**Source code:**

def calculate\_grade(exam\_score, assignment\_score, participation\_score):

exam\_weight = 0.4

assignment\_weight = 0.3

participation\_weight = 0.3

overall\_score = (exam\_score \* exam\_weight) + (assignment\_score \* assignment\_weight) + (participation\_score \* participation\_weight)

if exam\_score < 40 or assignment\_score < 40 or participation\_score == 0:

return "F"

elif overall\_score >= 90:

return "A"

elif overall\_score >= 80:

return "B"

elif overall\_score >= 70:

return "C"

elif overall\_score >= 60:

return "D"

else:

return "F"

exam\_score = float(input("Enter exam score: "))

assignment\_score = float(input("Enter assignment score: "))

participation\_score = float(input("Enter participation score: "))

# Calculate grade

final\_grade = calculate\_grade(exam\_score, assignment\_score, participation\_score)

print("Final Grade:", final\_grade)

**Sample Input and Output:**

Enter exam score: 90

Enter assignment score: 95

Enter participation score: 90

Final Grade: A

**Results:**

Thus, Final grade calculator using student ‘s exam score, assignment score and participation score have been implemented in python language and tested for various sample inputs.

**Ex No: 06 STRING MANIPULATION**

**Date: 07/02/2024**

**Aim:**

To write a Python program that performs string manipulation functions on the given string.

**Algorithm:**

1. Define a Python function named `title\_case` that takes a sentence as input.

2. Split the input sentence into a list of words using the `split()` method.

3. Iterate through each word in the list using list comprehension.

4. If the word is a common word (e.g., “and,” “the,” “in”), lowercase it unless it appears at the beginning of the sentence.

5. Otherwise, capitalize the first letter of the word and lowercase the rest.

6. Join the modified list of words back into a string using the `join()` method.

7. Return the resulting title-cased sentence.

**Source code:**

def title\_case(sentence):

common\_words = ["and", "the", "in"]

words = sentence.split()

title\_cased\_words = [word.capitalize() if i == 0 or word.lower() not in common\_words else word.lower() for i, word in enumerate(words)]

return ' '.join(title\_cased\_words)

input\_sentence = input("Enter your sentence: ")

output\_sentence = title\_case(input\_sentence)

print("Original Sentence:", input\_sentence)

print("Title Cased Sentence:", output\_sentence)

**Sample Input and Output:**

Enter your sentence: “the quick brown fox jumps over the lazy dog”

Original Sentence: the quick brown fox jumps over the lazy dog

Title Cased Sentence: The Quick Brown Fox Jumps Over the Lazy Dog

**Results:**

Thus, python program to implement different string manipulation techniques have been written successfully and tested with various samples.

**Ex No: 07 EXTRACTING TITLES USING LAMDA FUNCTION**

**Date: 14/02/2024**

**Aim:**

To write a Python program that sorts a list of books by year and extracts the books accordingly.

**Algorithm:**

1. Start

2. Define a list of dictionaries representing books, where each dictionary has ‘title’, ‘author’, and ‘year’ keys.

3. Use the sorted() function to sort the list of dictionaries based on the ‘year’ key in ascending order.

4. Use a lambda function with the map() function to create a new list containing only the titles of the books.

5. Return the sorted list of dictionaries and the list of book titles.

6. Stop.

**Source code:**

books = [

{'title': 'Book1', 'author': 'Author1', 'year': 2005},

{'title': 'Book2', 'author': 'Author2', 'year': 1998},

{'title': 'Book3', 'author': 'Author3', 'year': 2010},

{'title': 'Book4', 'author': 'Author4', 'year': 2000}

]

sorted\_books = sorted(books, key=lambda x: x['year'])

titles = list(map(lambda x: x['title'], sorted\_books))

print("Sorted Books:")

for book in sorted\_books:

print(book)

print("\nTitles of the Books:")

for title in titles:

print(title)

**Sample Input and Output:**

Sorted Books:

{'title': 'Book2', 'author': 'Author2', 'year': 1998}

{'title': 'Book4', 'author': 'Author4', 'year': 2000}

{'title': 'Book1', 'author': 'Author1', 'year': 2005}

{'title': 'Book3', 'author': 'Author3', 'year': 2010}

Titles of the Books:

Book2

Book4

Book1

Book3

**Result:**

Thus, the python program for sorting the books by using publishing year and also print only the sorted list of books was executed and verified successfully.

**Ex No: 08 STUDENT GRADE CALCULATOR WITH CLASS AND OBJECTS**

**Date: 14/02/2024**

**Aim:**

To create a Python program for managing student data, calculating their average marks, determining grades using classes and objects.

**Algorithm:**

1. Define a class named Student.

2. Initialize the class with attributes name, roll\_number, and marks.

3. Define a method calculate\_average() to calculate the average marks of the student.

4. Define a method get\_grade() to determine the grade based on the average marks calculated.

5. Define a method display\_info() to display the student's name, roll number, and average marks.

6. Create a dictionary student\_marks containing subject names as keys and marks as values.

7. Create an instance student1 of the Student class with name "Ragu", roll number "S001", and the dictionary of marks student\_marks.

8. Display the student's information using the display\_info() method.

9. Print the grade obtained by the student using the get\_grade() method.

**Source code:**

class Student:

def \_\_init\_\_(self, name, roll\_number, marks):

self.name = name

self.roll\_number = roll\_number

self.marks = marks

def calculate\_average(self):

total\_marks = sum(self.marks.values())

return total\_marks / len(self.marks)

def get\_grade(self):

average = self.calculate\_average()

if average >= 90:

return 'A'

elif 80 <= average < 90:

return 'B'

elif 70 <= average < 80:

return 'C'

elif 60 <= average < 70:

return 'D'

else:

return 'F'

def display\_info(self):

print("Student Name:", self.name)

print("Roll Number:", self.roll\_number)

print("Average Marks:", self.calculate\_average())

*# Example usage:*

student\_marks = {'Math': 85, 'Science': 90, 'History': 75}

student1 = Student("Ragu", "S001", student\_marks)

student1.display\_info()

print("Grade:", student1.get\_grade())

**Sample Input and Output:**

Student Name: Ragu

Roll Number: S001

Average Marks: 83.33333333333333

Grade: B

**Result:**

Thus, the Python program for managing student data, calculating their average marks, determining grades using classes and objects has been executed successfully.

**Ex No: 09 LIBRARY MANAGEMENT WITH CLASS AND OBJECTS**

**Date: 28/02/2024**

**Aim:**

To create a Python Program using class and objects where books can be added to a library, members can borrow and return books, and the status of books and members can be displayed.

**Algorithm:**

1. Define a class Book with attributes title, author, isbn, and availability.

2. Define a class Library with a list to store books and methods to add books and display book information.

3. Define a class Member with attributes member\_id, name, and a list to store borrowed books. It also includes methods to borrow and return books.

4. Define a class LibrarySystem which contains a library and a list of members. It includes methods to register members and display member information.

5. Create an instance of LibrarySystem.

6. Create instances of Book and add them to the library system's library.

7. Display the books in the library.

8. Create an instance of Member and register them with the library system.

9. Borrow a book for the member and display member information.

**Source code:**

class Book:

def \_\_init\_\_(self, title, author, isbn):

self.title = title

self.author = author

self.isbn = isbn

self.availability = True

class Library:

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

self.books = []

def add\_book(self, book):

self.books.append(book)

def display\_books(self):

for book in self.books:

print("Title:", book.title)

print("Author:", book.author)

print("ISBN:", book.isbn)

print("Availability:", "Available" if book.availability else "Not Available")

print()

class Member:

def \_\_init\_\_(self, member\_id, name):

self.member\_id = member\_id

self.name = name

self.borrowed\_books = []

def borrow\_book(self, book):

if book.availability:

self.borrowed\_books.append(book)

book.availability = False

print("Book", book.title, "borrowed successfully.")

else:

print("Book", book.title, "is not available for borrowing.")

def return\_book(self, book):

if book in self.borrowed\_books:

self.borrowed\_books.remove(book)

book.availability = True

print("Book", book.title, "returned successfully.")

else:

print("Book", book.title, "was not borrowed by this member.")

class LibrarySystem:

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

self.library = Library()

self.members = []

def register\_member(self, member):

self.members.append(member)

def display\_members(self):

for member in self.members:

print("Member ID:", member.member\_id)

print("Name:", member.name)

print("Borrowed Books:", [book.title for book in member.borrowed\_books])

print()

library\_system = LibrarySystem()

book1 = Book("Python Programming"," Guido van Rossum","978-0134444321")

book2 = Book("Internet of Things","Kalaiselvi Geetha"," 978-3-319-53470-1")

library\_system.library.add\_book(book1)

library\_system.library.add\_book(book2)

library\_system.library.display\_books()

member1 = Member("CS01", "Madhan")

library\_system.register\_member(member1)

member1.borrow\_book(book1)

library\_system.display\_members()

**Sample Input and Output:**

Title: Python Programming

Author: Guido van Rossum

ISBN: 978-0134444321

Availability: Available

Title: Internet of Things

Author: Kalaiselvi Geetha

ISBN: 978-3-319-53470-1

Availability: Available

Book Python Programming borrowed successfully.

Member ID: CS01

Name: Madhan

Borrowed Books: ['Python Programming']

**Result:**

Thus, the Python Program where books can be added to a library, members can borrow and return books, and the status of books and members can be displayed using class and objects has been executed successfully.

**Ex No: 10 SIMPLE BANK ACCOUNT USING OPERATOR OVERLOADING**

**Date: 13/03/2024**

**Aim:**

To Create a Python program to simulate a basic bank account system with deposit, withdrawal, and balance inquiry functionalities, along with operator overloading for account operations.

**Algorithm:**

1. Define a class BankAccount with attributes account\_number, account\_holder, and balance.

2. Include methods to deposit, withdraw, get\_balance, display\_account\_info, and overload operators for addition, subtraction, and equality.

3. Define the \_\_init\_\_ method to initialize the account with the account number, account holder's name, and initial balance.

4. Implement the deposit method to increase the balance by the deposited amount.

5. Implement the withdraw method to decrease the balance if sufficient funds are available.

6. Implement the get\_balance method to return the current balance.

7. Implement the display\_account\_info method to print the account information.

8. Overload the addition operator \_\_add\_\_ to combine balances of two accounts into a new account.

9. Overload the subtraction operator \_\_sub\_\_ to find the difference between balances of two accounts.

10. Overload the equality operator \_\_eq\_\_ to compare account numbers.

11. Create instances of BankAccount with different account details.

12. Test equality between two accounts.

13. Perform addition and subtraction operations between accounts and display the resulting account information.

**Source code:**

class BankAccount:

def \_\_init\_\_(self, account\_number, account\_holder, balance):

self.account\_number = account\_number

self.account\_holder = account\_holder

self.balance = balance

def deposit(self, amount):

self.balance += amount

def withdraw(self, amount):

if self.balance >= amount:

self.balance -= amount

print("Withdrawal successful. Current balance:", self.balance)

else:

print("Insufficient funds.")

def get\_balance(self):

return self.balance

def display\_account\_info(self):

print("Account Number:", self.account\_number)

print("Account Holder:", self.account\_holder)

print("Balance:", self.balance)

def \_\_add\_\_(self, other):

new\_balance = self.balance + other.balance

return BankAccount("Combined Account", "Joint Account", new\_balance)

def \_\_sub\_\_(self, other):

new\_balance = self.balance - other.balance

return BankAccount("Difference Account", "Difference Holder", new\_balance)

def \_\_eq\_\_(self, other):

return self.account\_number == other.account\_number

account1 = BankAccount("A001", "Kanthi", 1000)

account2 = BankAccount("A002", "Madhan", 500)

account3 = BankAccount("A001", "Pathy", 1500)

print(account1 == account2)

print(account1 == account3)

combined\_account = account1 + account2

combined\_account.display\_account\_info()

difference\_account = account1 - account2

difference\_account.display\_account\_info()

**Sample Input and Output:**

False

True

Account Number: Combined Account

Account Holder: Joint Account

Balance: 1500

Account Number: Difference Account

Account Holder: Difference Holder

Balance: 500

**Result:**

Thus, the Python program to simulate a basic bank account system with deposit, withdrawal, and balance inquiry functionalities, along with operator overloading for account operations has been executed successfully.

**Ex No: 11 INHERITANCE VEHICLE MANAGEMENT SYSTEM**

**Date: 13/03/2024**

**Aim:**

To create a Python program that models a transportation company's vehicle management system with a hierarchy of classes representing various vehicle types, allowing flexible management and polymorphic display of vehicle information.

**Algorithm:**

1. Define a class Vehicle with attributes make, model, year, and fuel\_type, and a method display\_info.

2. Define a class Car inheriting from Vehicle, with additional attributes num\_doors, num\_passengers, and car\_type, and override the display\_info method to include car-specific information.

3. Define a class Truck inheriting from Vehicle, with additional attributes payload\_capacity and four\_wheel\_drive, and override the display\_info method to include truck-specific information.

4. Define a class ElectricCar inheriting from Car, with additional attributes battery\_capacity and charging\_time, and override the display\_info method to include electric car-specific information.

5. Define a class Motorcycle inheriting from Vehicle, with additional attributes num\_wheels, has\_sidecar, and motorcycle\_type, and override the display\_info method to include motorcycle-specific information.

6. Define a function display\_vehicle\_info(vehicles) to display information for a list of vehicles, utilizing their display\_info methods.

7. Create instances of various vehicle types (Car, Truck, ElectricCar, Motorcycle).

8. Store these instances in a list.

9. Call display\_vehicle\_info function with the list of vehicles to print their information.

**Source code:**

class Vehicle:

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

self.make = make

self.model = model

self.year = year

self.fuel\_type = fuel\_type

def display\_info(self):

pass

class Car(Vehicle):

def \_\_init\_\_(self, make, model, year, fuel\_type, num\_doors, num\_passengers, car\_type):

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

self.num\_doors = num\_doors

self.num\_passengers = num\_passengers

self.car\_type = car\_type

def display\_info(self):

return f"Car: {self.make} {self.model} ({self.year}), Fuel: {self.fuel\_type}, Doors: {self.num\_doors}, Passengers: {self.num\_passengers}, Type: {self.car\_type}"

class Truck(Vehicle):

def \_\_init\_\_(self, make, model, year, fuel\_type, payload\_capacity, four\_wheel\_drive):

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

self.payload\_capacity = payload\_capacity

self.four\_wheel\_drive = four\_wheel\_drive

def display\_info(self):

return f"Truck: {self.make} {self.model} ({self.year}), Fuel: {self.fuel\_type}, Payload Capacity: {self.payload\_capacity}, 4WD: {self.four\_wheel\_drive}"

class ElectricCar(Car):

def \_\_init\_\_(self, make, model, year, num\_doors, num\_passengers, car\_type, battery\_capacity, charging\_time):

super().\_\_init\_\_(make, model, year, "Electric", num\_doors, num\_passengers, car\_type)

self.battery\_capacity = battery\_capacity

self.charging\_time = charging\_time

def display\_info(self):

return f"Electric Car: {self.make} {self.model} ({self.year}), Battery Capacity: {self.battery\_capacity}, Charging Time: {self.charging\_time}, {super().display\_info()}"

class Motorcycle(Vehicle):

def \_\_init\_\_(self, make, model, year, fuel\_type, num\_wheels, has\_sidecar, motorcycle\_type):

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

self.num\_wheels = num\_wheels

self.has\_sidecar = has\_sidecar

self.motorcycle\_type = motorcycle\_type

def display\_info(self):

return f"Motorcycle: {self.make} {self.model} ({self.year}), Fuel: {self.fuel\_type}, Wheels: {self.num\_wheels}, Sidecar: {self.has\_sidecar}, Type: {self.motorcycle\_type}"

def display\_vehicle\_info(vehicles):

for vehicle in vehicles:

print(vehicle.display\_info())

car1 = Car("Toyota", "Camry", 2022, "Gasoline", 4, 5, "Sedan")

truck1 = Truck("Ford", "F-150", 2022, "Gasoline", 1500, True)

electric\_car1 = ElectricCar("Tesla", "Model S", 2022, 4, 5, "Sedan", 100, 8)

motorcycle1 = Motorcycle("Harley-Davidson", "Sportster", 2022, "Gasoline", 2, False, "Cruiser")

vehicles\_list = [car1, truck1, electric\_car1, motorcycle1]

display\_vehicle\_info(vehicles\_list)

**Sample Input and Output:**

Car: Toyota Camry (2022), Fuel: Gasoline, Doors: 4, Passengers: 5, Type: Sedan

Truck: Ford F-150 (2022), Fuel: Gasoline, Payload Capacity: 1500, 4WD: True

Electric Car: Tesla Model S (2022), Battery Capacity: 100, Charging Time: 8, Car: Tesla Model S (2022), Fuel: Electric, Doors: 4, Passengers: 5, Type: Sedan

Motorcycle: Harley-Davidson Sportster (2022), Fuel: Gasoline, Wheels: 2, Sidecar: False, Type: Cruiser

**Result:**

Thus, the Python program that models a transportation company's vehicle management system with a hierarchy of classes representing various vehicle types, allowing flexible management and polymorphic display of vehicle information has been executed successfully.

**Ex No: 12 FILE HANDLING ON LOG FILES**

**Date: 20/03/2024**

**Aim:**

To read and analyze a log file, extracting timestamped entries to determine the total number of entries, count occurrences of each severity level, and calculate the average time gap between consecutive log entries, facilitating effective log data assessment and system monitoring using python.

**Algorithm:**

1. Reading the log file:

* Opens a log file and reads its contents line by line into a list.

1. Extracting information from log entries:

* Defines a pattern to extract timestamp, severity, and message from each log entry using regular expressions.
* Parses each log entry to extract this information and converts the timestamp into a datetime object.

1. Analyzing the log entries:

* Counts the total number of log entries.
* Counts the occurrences of different severity levels.
* Calculates the time gap between consecutive log entries and computes the average time gap.

1. Main Functionality:

* Runs the main code if the script is executed directly.
* Calls functions to read the log file, analyze its contents, and print the results.

**Source code:**

import re

from datetime import datetime, timedelta

def read\_log\_file(file\_path):

with open(file\_path, 'r') as file:

log\_entries = file.readlines()

return log\_entries

def extract\_information(log\_entry):

*# Define a regular expression pattern to extract timestamp, severity, and message*

pattern = r'(\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}) - (\w+): (.\*)'

match = re.match(pattern, log\_entry)

if match:

timestamp\_str, severity, message = match.groups()

timestamp = datetime.strptime(timestamp\_str, '%Y-%m-%d %H:%M:%S')

return timestamp, severity, message

else:

return None

def analyze\_log(log\_entries):

total\_entries = len(log\_entries)

severity\_counts = {}

time\_gaps = []

for i in range(1, total\_entries):

current\_entry = extract\_information(log\_entries[i])

previous\_entry = extract\_information(log\_entries[i - 1])

if current\_entry and previous\_entry:

time\_gap = current\_entry[0] - previous\_entry[0]

time\_gaps.append(time\_gap.total\_seconds())

*# Count severity levels*

severity\_counts[current\_entry[1]] = severity\_counts.get(current\_entry[1], 0) + 1

average\_time\_gap = sum(time\_gaps) / len(time\_gaps) if time\_gaps else 0

return total\_entries, severity\_counts, average\_time\_gap

if \_\_name\_\_ == "\_\_main\_\_":

log\_file\_path = "log\_file.log"

log\_entries = read\_log\_file(log\_file\_path)

total\_entries, severity\_counts, average\_time\_gap = analyze\_log(log\_entries)

print(f"Total Entries: {total\_entries}")

print("Severity Counts:")

for severity, count in severity\_counts.items():

print(f" {severity}: {count}")

print(f"Average Time Gap between Entries: {average\_time\_gap} seconds")

**Source Code (Log File):**

***Note: Save a Log File as log\_file.log***

2024-02-27 10:00:00 - INFO: Application started

2024-02-27 10:05:30 - ERROR: Critical error occurred - Server crashed

2024-02-27 10:10:45 - WARNING: Resource usage high

2024-02-27 10:15:20 - INFO: User logged in

2024-02-27 10:20:05 - DEBUG: Debugging message - Step 1

***Note: Create and save LOG file and change the directory name to the LOG file directory which you created in the python source code and execute.***

**Sample Input and Output:**

Total Entries: 5

Severity Counts:

ERROR: 1

WARNING: 1

INFO: 1

DEBUG: 1

Average Time Gap between Entries: 301.25 seconds

**Result:**

Thus, the log analysis program executed successfully, determining the total entries, average time gap, and severity level counts. It provided concise insights into the log data.

**Ex No: 13 FILE HANDLING ON CSV FILES**

**Date: 20/03/2024**

**Aim:**

To Design a Python program to efficiently handle and analyze employee data stored in a CSV file, reading the file, finding the highest-paid employee, sorting employees by department, and calculating the average salary for each department.

**Algorithm:**

1. Import Libraries:

- Import `csv` and `operator` for CSV handling and sorting.

2. Read CSV:

- Use `csv.reader` to read the employee data from the CSV file.

3. Highest-Paid Employee:

- Track the highest-paid employee while iterating through the data.

4. Sort Employees by Department:

- Utilize the `sorted` function to sort employees based on department.

5. Average Salary per Department:

- Calculate the average salary for each department using a dictionary.

6. Display Results:

- Print the details of the highest-paid employee, the sorted employee list, and the average salary for each department.

7. Exception Handling:

- Implement basic error handling for file reading or data processing issues.

8. Close File:

- Ensure proper closure of the CSV file.

**Source code:**

import csv

from collections import defaultdict

def read\_csv\_file(file\_path):

employees = []

with open(file\_path, 'r') as file:

reader = csv.DictReader(file)

for row in reader:

employees.append(row)

return employees

def find\_highest\_paid\_employee(employees):

highest\_paid\_employee = max(employees, key=lambda x: float(x['salary']))

return highest\_paid\_employee

def sort\_employees\_by\_department(employees):

sorted\_employees = sorted(employees, key=lambda x: x['department'])

return sorted\_employees

def calculate\_average\_salary\_by\_department(employees):

department\_salaries = defaultdict(list)

for employee in employees:

department\_salaries[employee['department']].append(float(employee['salary']))

average\_salaries = {department: sum(salaries) / len(salaries) for department, salaries in department\_salaries.items()}

return average\_salaries

def main():

file\_path = 'emp.csv'

employees = read\_csv\_file(file\_path)

highest\_paid\_employee = find\_highest\_paid\_employee(employees)

print(f"Highest Paid Employee: {highest\_paid\_employee['name']} (ID: {highest\_paid\_employee['employee\_id']}, Salary: {highest\_paid\_employee['salary']})")

sorted\_employees = sort\_employees\_by\_department(employees)

print("\nEmployees Sorted by Department:")

for employee in sorted\_employees:

print(f"{employee['name']} (ID: {employee['employee\_id']}, Department: {employee['department']}, Salary: {employee['salary']})")

average\_salaries = calculate\_average\_salary\_by\_department(employees)

print("\nAverage Salary by Department:")

for department, avg\_salary in average\_salaries.items():

print(f"{department}: {avg\_salary:.2f}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Source Code (CSV File):**

***Note: Save a CSV File as employee\_data.csv***

employee\_id,name,department,salary

1,Sriram ,HR,50000

2,Vasanth,IT,60000

3,Praneeth,HR,55000

4,Suresh,IT,65000

5,Ramesh,Finance,70000

***Note: Create and save CSV file and change the directory name to the CSV file directory which you created in the python source code and execute.***

**Sample Input and Output:**

Highest Paid Employee: Ramesh (ID: 5, Salary: 70000)

Employees Sorted by Department:

Ramesh (ID: 5, Department: Finance, Salary: 70000)

Sriram (ID: 1, Department: HR, Salary: 50000)

Praneeth (ID: 3, Department: HR, Salary: 55000)

Vasanth (ID: 2, Department: IT, Salary: 60000)

Suresh (ID: 4, Department: IT, Salary: 65000)

Average Salary by Department:

HR: 52500.00

IT: 62500.00

Finance: 70000.00

**Result:**

Thus, the File handling on csv file is executed successfully, calculating the highest-paid employee, sorting employees by department, and calculating the average salary for each department.

**Ex No: 14 CALCULATOR USING EXCEPTION HANDLING**

**Date: 27/03/2024**

**Aim:**

To Develop a user-friendly calculator program with exception handling to ensure error-free input for basic arithmetic operations, including informative error messages for potential issues.

**Algorithm:**

1. The calculate function:

* It takes three parameters: two numbers (num1 and num2) and an operation (operation).
* It performs the specified operation (+, -, \*, /) on the numbers and returns the result.
* It includes error handling for division by zero, invalid operations, and invalid input types.

1. The get\_user\_input function:

* It prompts the user to enter two numbers and an operation.
* It converts the user input into floating-point numbers.
* It includes error handling for invalid input types.

1. The main part of the script:

* It runs a loop to continuously prompt the user for input and perform calculations.
* It calls the get\_user\_input function to get user input.
* It calls the calculate function to perform the calculation and prints the result.
* It asks the user if they want to continue, and if not, it breaks the loop.

**Source code:**

def calculate(num1, num2, operation):

try:

if operation == '+':

result = num1 + num2

elif operation == '-':

result = num1 - num2

elif operation == '\*':

result = num1 \* num2

elif operation == '/':

if num2 == 0:

raise ZeroDivisionError("Cannot divide by zero")

result = num1 / num2

else:

raise ValueError("Invalid operation. Please use '+', '-', '\*', or '/'.")

return result

except (ValueError, TypeError) as e:

print(f"Error: {e}")

except ZeroDivisionError as e:

print(f"Error: {e}")

def get\_user\_input():

try:

num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

operation = input("Enter the operation (+, -, \*, /): ")

return num1, num2, operation

except ValueError:

print("Error: Invalid input for numbers.")

return None, None, None

if \_\_name\_\_ == "\_\_main\_\_":

while True:

num1, num2, operation = get\_user\_input()

if num1 is not None and num2 is not None and operation is not None:

result = calculate(num1, num2, operation)

if result is not None:

print(f"Result: {result}")

user\_input = input("Do you want to continue? (y/n): ").lower()

if user\_input != 'y':

break

**Sample Input and Output:**

Enter the first number: 6

Enter the second number: 9

Enter the operation (+, -, \*, /): +

Result: 15.0

Do you want to continue? (y/n): n

**Result:**

Thus, the arithmetic calculator program executed successfully. Users can perform basic arithmetic operations on two numbers. The program handles invalid input types gracefully, ensuring a smooth user experience.

**Ex No: 15 NUMERICAL DATA PROCESSING USING PANDAS**

**Date: 27/03/2024**

**Aim:**

To create a Python program that reads numerical data from a file, performs calculations, and handles potential errors gracefully.

**Algorithm:**

1. Define a function to read numerical data from a file.
2. Open the specified file, iterating through each line.
3. Attempt to convert each line to a float, appending valid values to a list.
4. Handle potential errors, such as a missing file or invalid data formats.
5. Perform numerical calculations on the collected data.
6. Handle potential errors during calculations, like division by zero.
7. Print the total and average if calculations are successful.
8. Provide clear feedback to the user throughout the process.

**Source code:**

def read\_data\_from\_file(file\_path):

data = []

try:

with open(file\_path, 'r') as file:

for line in file:

try:

data.append(float(line.strip()))

except ValueError:

print(f"Ignoring non-numeric data: {line.strip()}")

except FileNotFoundError:

print(f"File '{file\_path}' not found.")

except Exception as e:

print(f"An error occurred while reading the file: {e}")

return data

def perform\_numerical\_calculations(data):

try:

if not data:

raise ValueError("No numerical data found.")

total = sum(data)

average = total / len(data)

return total, average

except ZeroDivisionError:

print("Cannot calculate average: Division by zero.")

except Exception as e:

print(f" An error occurred during numerical calculations: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

file\_path = 'data.txt'

data = read\_data\_from\_file(file\_path)

total, average = perform\_numerical\_calculations(data)

if total is not None and average is not None:

print(f"Total: {total}")

print(f"Average: {average}")

**Source Code (TXT File):**

***Note: Save a TXT File as data.txt***

10

20

30

40

50

abc

60

70

80

***Note: Create and save TXT file and change the directory name to the TXT file directory which you created in the python source code and execute.***

**Sample Input and Output:**

Ignoring non-numeric data: abc

Total: 360.0

Average: 45.0

**Result:**

Thus, the Python program for numerical data analysis and error handling has been executed successfully.

**Ex No: 16 E-COMMERCE SALES ANALYSIS WITH MATPLOTLIB**

**Date: 03/04/2024**

**Aim:**

To create a Python program for analyzing sales transactions dataset, including data loading, exploration, cleaning, manipulation, visualization, and advanced analysis using NumPy, Pandas, and Matplotlib.

**Algorithm:**

1) Import Libraries:

import pandas (pd), numpy (np), and matplotlib.pyplot (plt).

2) Load and Explore Data:

* Load the dataset into a DataFrame (df) using pd.read\_csv().
* Print descriptive statistics with df.describe() and display the first few rows with df.head().

3) Data Cleaning and Manipulation:

* Check for missing values with df.isnull().sum().
* Convert 'Date' column to datetime format using pd.to\_datetime().
* Calculate 'Total\_Price' by multiplying 'Quantity' and 'Price\_per\_Unit'.

4) Data Visualization:

* Group data by 'Product\_Name' and plot total sales for each product as a bar chart.
* Group data by 'Date' and plot sales trend over time as a line chart.
* Create a scatter plot to visualize the relationship between 'Quantity' and 'Total\_Price'.

5) Advanced Analysis:

* Calculate correlation coefficient between 'Quantity' and 'Total\_Price' using np.corrcoef().
* Find average spending per customer by grouping data by 'Customer\_ID' and calculating mean 'Total\_Price'.
* Identify top 5 products based on total sales using product\_sales.nlargest(5).

6) Display Visualization:

Use plt.show() to display each plot.

**Source code:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

*# Data Loading and Exploration*

df = pd.read\_csv('sales\_data.csv')

print("Descriptive Statistics using NumPy:")

print(np.array(df.describe()))

print("\nDescriptive Statistics using Pandas:")

print(df.describe())

print("\nFirst few rows of the dataset:")

print(df.head())

*# Data Cleaning and Manipulation*

print("\nMissing values:")

print(df.isnull().sum())

df['Date'] = pd.to\_datetime(df['Date'])

df['Total\_Price'] = df['Quantity'] \* df['Price\_per\_Unit']

*# Data Visualization*

product\_sales = df.groupby('Product\_Name')['Total\_Price'].sum()

plt.figure(figsize=(10, 6))

product\_sales.plot(kind='bar', color='red')

plt.title('Total Sales for Each Product')

plt.xlabel('Product')

plt.ylabel('Total Sales ($)')

plt.xticks(rotation=45)

plt.show()

plt.figure(figsize=(10, 6))

sales\_trend = df.groupby('Date')['Total\_Price'].sum()

sales\_trend.plot(kind='line', marker='o', color='orange')

plt.title('Sales Trend Over Time')

plt.xlabel('Date')

plt.ylabel('Total Sales ($)')

plt.show()

plt.figure(figsize=(10, 6))

plt.scatter(df['Quantity'], df['Total\_Price'], color='green')

plt.title('Relationship between Quantity and Total Price')

plt.xlabel('Quantity')

plt.ylabel('Total Price ($)')

plt.show()

*# Advanced Analysis*

correlation\_coefficient = np.corrcoef(df['Quantity'], df['Total\_Price'])[0,1]

print("\nCorrelation Coefficient between Quantity and Total Price:", correlation\_coefficient)

average\_spending\_per\_customer = df.groupby('Customer\_ID')['Total\_Price'].mean()

print("\nAverage Total Spending per Customer:")

print(average\_spending\_per\_customer)

top\_5\_products = product\_sales.nlargest(5)

plt.figure(figsize=(10, 6))

top\_5\_products.plot(kind='bar', color='purple')

plt.title('Top 5 Products by Total Sales')

plt.xlabel('Product')

plt.ylabel('Total Sales ($)')

plt.xticks(rotation=45)

plt.show()

**Source Code (TXT File):**

***Note: Save a CSV File as sales\_data.csv***

Transaction\_ID,Product\_Name,Quantity,Price\_per\_Unit,Customer\_ID,Date

1,Shoes,2,50,101,2023-01-01

2,T-shirt,3,20,102,2023-01-02

3,Jeans,1,80,103,2023-01-03

4,Shoes,2,50,104,2023-01-04

5,T-shirt,2,20,101,2023-01-05

6,Jeans,4,80,102,2023-01-06

7,Shoes,1,50,103,2023-01-07

8,T-shirt,5,20,104,2023-01-08

9,Jeans,2,80,101,2023-01-09

10,Shoes,3,50,102,2023-01-10

***Note: Create and save CSV file and change the directory name to the CSV file directory which you created in the python source code and execute.***

**Sample Input and Output:**

Descriptive Statistics using NumPy:

[[ 10. 10. 10. 10. ]

[ 5.5 2.5 50. 102.3 ]

[ 3.02765035 1.26929552 24.49489743 1.15950181]

[ 1. 1. 20. 101. ]

[ 3.25 2. 27.5 101.25 ]

[ 5.5 2. 50. 102. ]

[ 7.75 3. 72.5 103. ]

[ 10. 5. 80. 104. ]]

Descriptive Statistics using Pandas:

Transaction\_ID Quantity Price\_per\_Unit Customer\_ID

count 10.00000 10.000000 10.000000 10.000000

mean 5.50000 2.500000 50.000000 102.300000

std 3.02765 1.269296 24.494897 1.159502

min 1.00000 1.000000 20.000000 101.000000

25% 3.25000 2.000000 27.500000 101.250000

50% 5.50000 2.000000 50.000000 102.000000

75% 7.75000 3.000000 72.500000 103.000000

max 10.00000 5.000000 80.000000 104.000000

First few rows of the dataset:

Transaction\_ID Product\_Name Quantity Price\_per\_Unit Customer\_ID Date

0 1 Shoes 2 50 101 2023-01-01

1 2 T-shirt 3 20 102 2023-01-02

2 3 Jeans 1 80 103 2023-01-03

3 4 Shoes 2 50 104 2023-01-04

4 5 T-shirt 2 20 101 2023-01-05

Missing values:

Transaction\_ID 0

Product\_Name 0

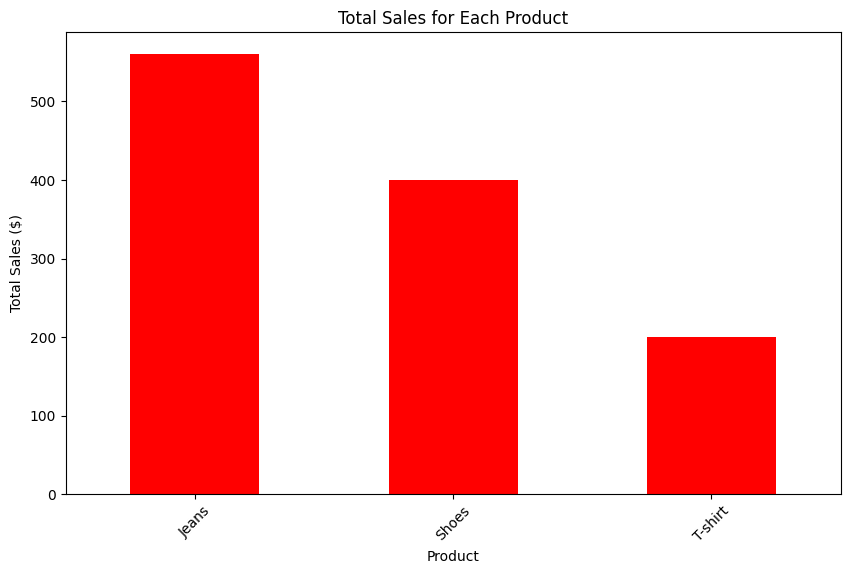
Quantity 0

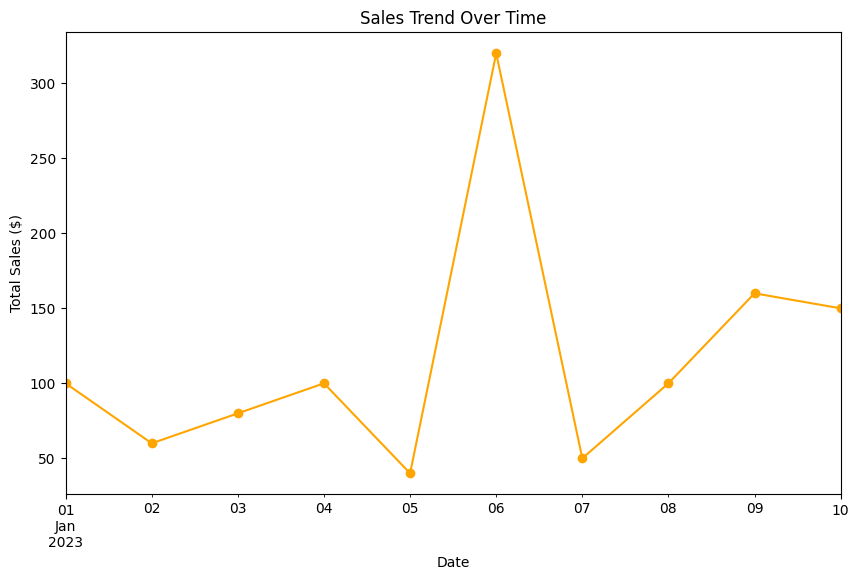
Price\_per\_Unit 0

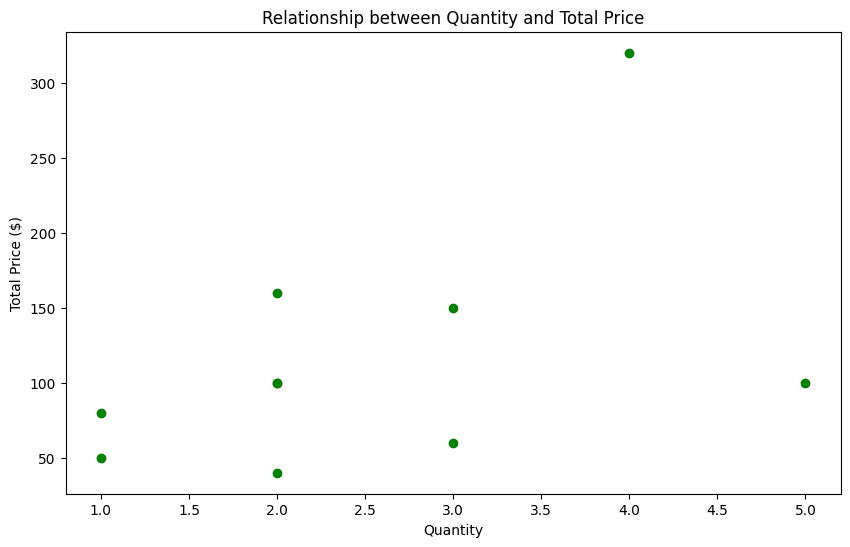
Customer\_ID 0

Date 0

dtype: int64







Correlation Coefficient between Quantity and Total Price: 0.4715723507347863

Average Total Spending per Customer:

Customer\_ID

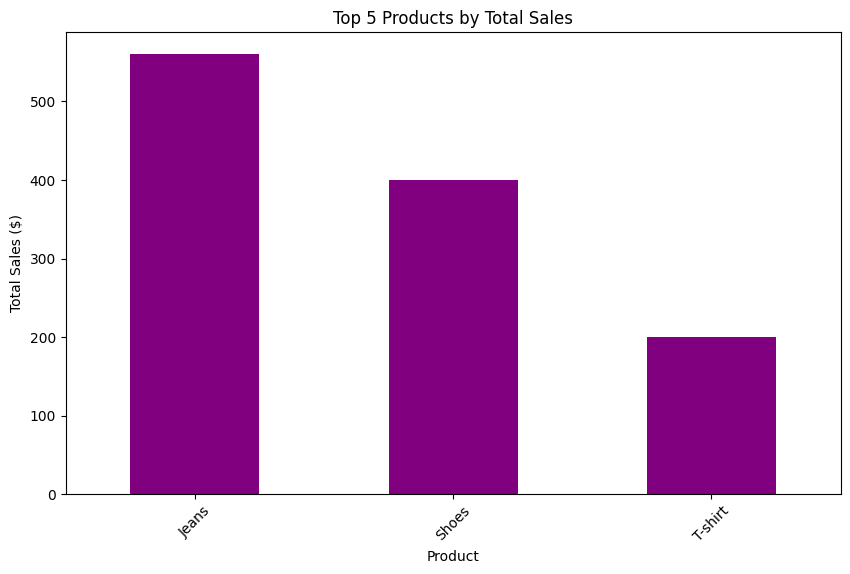
101 100.000000

102 176.666667

103 65.000000

104 100.000000

Name: Total\_Price, dtype: float64



**Result:**

Thus, the Python program for analyzing sales transactions dataset, including data loading, exploration, cleaning, manipulation, visualization, and advanced analysis using NumPy, Pandas, and Matplotlib has been executed successfully.

**Ex No: 17 TIC-TAC-TOE GAME WITH TKINTER**

**Date: 03/04/2024**

**Aim:**

To create a Python program using Tkinter for a two-player Tic-Tac-Toe game.

**Algorithm:**

1. Import the necessary modules: tkinter and messagebox.

2. Define a class TicTacToe to represent the game.

3. Initialize the game attributes such as the Tkinter root window, current player, game board, and buttons grid in the constructor (\_\_init\_\_ method).

4. Create a method create\_board() to generate the game board with buttons.

5. Implement the make\_move() method to handle player moves, update the board, and check for a winner or draw.

6. Define the check\_winner() method to verify winning conditions by checking rows, columns, and diagonals.

7. Implement highlight\_winner() method to visually highlight the winning combination on the GUI.

8. Implement check\_draw() method to check for a draw condition.

9. Define end\_game() method to display the result (winner or draw) using a messagebox and quit the game.

10. Add a play() method to start the main event loop using root.mainloop().

11. In the main block, create an instance of TicTacToe, and call its play() method to start the game loop.

**Source code:**

import tkinter as tk

from tkinter import messagebox

class TicTacToe:

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

self.root = tk.Tk()

self.root.title("Tic-Tac-Toe")

self.current\_player = "X"

self.board = [[' ' for \_ in range(3)] for \_ in range(3)]

self.buttons = [[None for \_ in range(3)] for \_ in range(3)]

self.create\_board()

def create\_board(self):

for i in range(3):

for j in range(3):

self.buttons[i][j] = tk.Button(self.root, text="", font=("Helvetica", 20), width=5, height=2,command=lambda row=i, col=j: self.make\_move(row, col))

self.buttons[i][j].grid(row=i, column=j)

def make\_move(self, row, col):

if self.board[row][col] == ' ':

self.board[row][col] = self.current\_player

self.buttons[row][col].config(text=self.current\_player)

if self.check\_winner() or self.check\_draw():

self.end\_game()

else:

self.current\_player = 'O' if self.current\_player == 'X' else 'X'

def check\_winner(self):

for i in range(3):

if self.board[i][0] == self.board[i][1] == self.board[i][2] != ' ':

self.highlight\_winner(i, 0, i, 1, i, 2)

return True

if self.board[0][i] == self.board[1][i] == self.board[2][i] != ' ':

self.highlight\_winner(0, i, 1, i, 2, i)

return True

if self.board[0][0] == self.board[1][1] == self.board[2][2] != ' ':

self.highlight\_winner(0, 0, 1, 1, 2, 2)

return True

if self.board[0][2] == self.board[1][1] == self.board[2][0] != ' ':

self.highlight\_winner(0, 2, 1, 1, 2, 0)

return True

return False

def highlight\_winner(self, \*coords):

for i in range(0, len(coords), 2):

self.buttons[coords[i]][coords[i+1]].config(bg='light green')

def check\_draw(self):

for row in self.board:

for cell in row:

if cell == ' ':

return False

return True

def end\_game(self):

if self.check\_winner():

messagebox.showinfo("Winner", f"Player {self.current\_player} wins!")

else:

messagebox.showinfo("Draw", "It's a draw!")

self.root.quit()

def play(self):

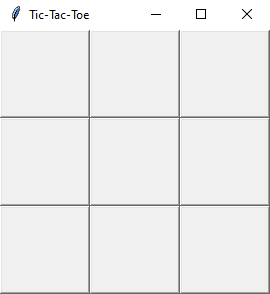
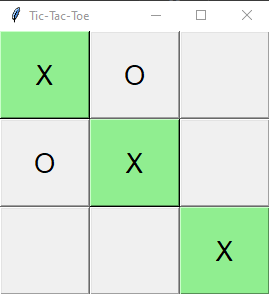
self.root.mainloop()

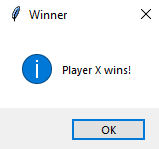
if \_\_name\_\_ == "\_\_main\_\_":

game = TicTacToe()

game.play()

**Sample Input and Output:**



**Result:**

Thus, the Python program using Tkinter for a two-player Tic-Tac-Toe game has been executed successfully.