**PYTHON LAB CYCLE**

**Dilsha C P**

**MCA2318**

**1. Python Programming**

|  |
| --- |
| **1.1. Create a simple calculator in Python.** |

**PROGRAM:**

|  |
| --- |
| #Simple Calculator  num1 = float(input('Enter the first Number : '))  num2 = float(input('Enter the second Number : '))  op = input('Select the operator : \n+\n-\n\*\n/\n')  if op == '+' :      sum = num1 + num2      print('Sum of two numbers is ',sum)  elif op == '-' :      diff = num1 - num2      print('Difference of two numbers is ',diff)  elif op == '\*' :      prod = num1 \* num2      print('Product of two numbers is : ',prod)  elif op == '/' :      if num2 == 0 :          print('Division by zero Not possible')      div = num1 / num2      print('Division : ',div)  else :      print('Invalid input') |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.2. An electric power distribution company charges domestic customers as**  **follows: Consumption unit Rate of charge:**  **1.2.1. 0-200 Rs. 0.50 per unit**  **1.2.2. 201-400 Rs. 0.65 per unit in excess of 200**  **1.2.3. 401-600 Rs 0.80 per unit excess of 400**  **1.2.4. 601 and above Rs 1.00per unit excess of 600**  **1.2.5. If the bill exceeds Rs. 400, then a surcharge of 15% will be charged,**  **and the minimum bill should be Rs. 100/-**  **Create a Python program based on the scenario mentioned above.** |

**PROGRAM:**

|  |
| --- |
| #Function to calculate Bill  def billCalculator(cUnits):      if cUnits <= 200:          bill = cUnits \* 0.50      elif cUnits <= 400:          bill = (200 \* 0.50) + (cUnits -200) \* 0.65      elif cUnits <= 600:          bill = (200 \* 0.50) + (200 \* 0.65) + (cUnits - 400) \* 0.80      else:          bill = (200 \* 0.50) + (200 \* 0.65) + (200 \* 0.80) + (cUnits - 600) \* 1.00        #Add surcharge to bill if bill amount > 400      if bill > 400 :          surCharge = (bill - 400) \* 0.15          bill += surCharge        #If bill < 100 Make bill as 100      if bill < 100 :          bill = 100      return bill  print("\nProgram to calculate electric power consumption -->\n")  consumedUnits = float(input("\nEnter the units consumed : "))  totalBill = billCalculator(consumedUnits)  print(f"Total Bill : Rs. {totalBill}") |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.3. Print the pyramid of numbers using for loops.** |

**PROGRAM:**

|  |
| --- |
| #Program to display number Pyramid  def numPyramid(n):      for i in range(1 , n + 1):          for j in range( n - i):              print(" ",end ="") #To print leading spaces            for j in range(1 , i+1):              print(j ,end="") #printing num in ascending order            for j in range (i - 1 ,0 , -1):              print(j , end="") #print in desc order            print() # To move to next line      n = int(input("Enter how many no.of rows to display : "))  numPyramid(n) |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.4. Write a program to find the number and sum of all integers greater than 100 and less than 200 that are divisible by 7.** |

**PROGRAM:**

|  |
| --- |
| #Program to find the number and  #Sum of all num > 100 and < 200 that are divisible by 7  def findNumAndSum():      count = 0      total\_sum = 0        for num in range(101, 200):          if num % 7 == 0:              count += 1              total\_sum += num        return count, total\_sum  count, totalSum = findNumAndSum()  print(f"The number of integers greater than 100 and less than 200 that are divisible by 7 is: {count}")  print(f"The sum of all these integers is: {totalSum}") |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.5. Write a recursive function to calculate the sum of numbers from 0 to 10** |

**PROGRAM :**

|  |
| --- |
| #Program to calculate the sum of numbers from 0 to 10 using recursion  def recursiveSum(n):      # Base case      if n == 0:          return 0      else:          # Recursive case          return n + recursiveSum(n - 1)  result = recursiveSum(10)  print(f"The sum of numbers from 0 to 10 is: {result}") |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.6. Write a Python program to reverse the digits of a given number and add them to the original. If the sum is not a palindrome, repeat this procedure.** |

**PROGRAM :**

|  |
| --- |
| #Write a Python program to reverse the digits of a given number and add them  #to the original. If the sum is not a palindrome, repeat this procedure.  def isPalindrome(n):      original = n      reverse = 0      while n > 0:          digit = n % 10          reverse = reverse \* 10 + digit          n = n // 10      return original == reverse  def reverseNumber(n):      reverse = 0      while n > 0:          digit = n % 10          reverse = reverse \* 10 + digit          n = n // 10 # It returns the quotient which is rounded down to the nearest integer.      return reverse  def reverseAndAddUntilPalindrome(n):      while not isPalindrome(n):          reversed\_n = reverseNumber(n)          n = n + reversed\_n          print(f"Reversed: {reversed\_n}, Sum: {n}")      return n  number = int(input("Enter a number: "))  result = reverseAndAddUntilPalindrome(number)  print(f"The resulting palindrome is: {result}") |

**OUTPUT :**

|  |
| --- |
|  |

|  |
| --- |
| **1.7. Write a menu-driven program that performs the following operations on**  **strings**  **1.7.1. Check if the String is a Substring of Another String**  **1.7.2. Count Occurrences of Character**  **1.7.3. Replace a substring with another substring**  **1.7.4. Convert to Capital Letters** |

**PROGRAM :**

|  |
| --- |
| #Write a menu-driven program that performs the following operations on  # strings  # 1. Check if the String is a Substring of Another String  # 2. Count Occurrences of Character  # 3. Replace a substring with another substring  # 4. Convert to Capital Letters  def checkSubstring():      string = input("Enter the main string: ")      substring = input("Enter the substring to check: ")        if substring in string:          print(f"'{substring}' is a substring of '{string}'")      else:          print(f"'{substring}' is not a substring of '{string}'")  def countOccurrences():      string = input("Enter the string: ")      char = input("Enter the character to count: ")        count = string.count(char)      print(f"Number of occurrences of '{char}' in '{string}': {count}")  def replaceSubstring():      string = input("Enter the main string: ")      old\_substring = input("Enter the substring to replace: ")      new\_substring = input("Enter the new substring: ")        new\_string = string.replace(old\_substring, new\_substring)      print(f"Modified string: '{new\_string}'")  def convertToUpper():      string = input("Enter the string to convert to uppercase: ")      uppercase\_string = string.upper()      print(f"Uppercase string: '{uppercase\_string}'")  # Main program  while True:      print("\nMenu:")      print("1. Check if String is a Substring of Another String")      print("2. Count Occurrences of Character")      print("3. Replace a substring with another substring")      print("4. Convert to Capital Letters")      print("5. Exit")        choice = input("Enter your choice (1-5): ")        if choice == '1':          checkSubstring()      elif choice == '2':          countOccurrences()      elif choice == '3':          replaceSubstring()      elif choice == '4':          convertToUpper()      elif choice == '5':          print("Exiting the program...")          break      else:          print("Invalid choice! Please enter a number from 1 to 5.") |

**OUTPUT :**

|  |
| --- |
|  |

|  |
| --- |
| **1.8. Write a function to find the factorial of a number but also store the factorials calculated in a dictionary.** |

**PROGRAM:**

|  |
| --- |
| #1.8. Write a function to find the factorial of a number but also store the factorials calculated in a dictionary.  def factorial\_with\_cache(n, cache={}):      if n < 0:          raise ValueError("Factorial is not defined for negative numbers.")      if n in cache:          return cache[n]      if n == 0 or n == 1:          result = 1      else:          result = n \* factorial\_with\_cache(n - 1, cache)      cache[n] = result      print(cache[n])      return result  number = int(input("Enter a number to calculate its factorial: "))  factorial = factorial\_with\_cache(number)  print(f"The factorial of {number} is: {factorial}") |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.9. Perform various set operations**  **1.9.1. Set Union**  **1.9.2. Set Intersection**  **1.9.3. Set Difference** |

**PROGRAM :**

|  |
| --- |
| #Perform various set operations  # 1. Set Union  # 2. Set Intersection  # 3. Set Difference  def setOperations(set1, set2):        union\_set = set1.union(set2)        intersection\_set = set1.intersection(set2)        difference\_set1 = set1.difference(set2)        difference\_set2 = set2.difference(set1)        return union\_set, intersection\_set, difference\_set1, difference\_set2  set1 = {1, 2, 3, 4, 5}  set2 = {3, 4, 5, 6, 7}  union\_set, intersection\_set, difference\_set1, difference\_set2 = setOperations(set1, set2)  print(f"Set1: {set1}")  print(f"Set2: {set2}")  print(f"Union of Set1 and Set2: {union\_set}")  print(f"Intersection of Set1 and Set2: {intersection\_set}")  print(f"Difference (Set1 - Set2): {difference\_set1}")  print(f"Difference (Set2 - Set1): {difference\_set2}") |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.10. Create a dictionary to store the name, roll\_no, and total\_mark of N students.**  **Now print the details of the student with the highest total\_mark.** |

**PROGRAM :**

|  |
| --- |
| # Create a dictionary to store the name, roll\_no, and total\_mark of N students.  #Now print the details of the student with the highest total\_mark.  #lambda x: x['marks'] is a lambda function defined inline.  #  It takes an argument x (which represents each student dictionary in students) and returns x['marks'].  # This lambda function extracts the 'marks' value from each student dictionary.  def student\_details(N):      student=[]      for i in range(N):          print("\nEnter details of Student ",i+1," : ")          name = input("Enter student name: ")          roll\_no=input("Enter the roll no:")          marks = int(input("Enter the marks: "))          students={              'name':name,              'roll\_no':roll\_no,              'marks':marks          }          student.append(students)      return student  def find\_top\_student(students):      top\_student = max(students, key=lambda x: x['marks'])      return top\_student  N=int(input("Enter the number of students:"))  Student=student\_details(N)  print("Student Details:\n")  print(Student)  top\_student=find\_top\_student(Student)  print("Details of Top Student:\n")  print("Name:",top\_student['name'])  print("Roll No:",top\_student['roll\_no'])  print("Marks:",top\_student['marks']) |

**OUTPUT :**

|  |
| --- |
|  |

|  |
| --- |
| **1.11. Write a Python program to copy the contents of a file into another file, line by line.** |

**PROGRAM :**

|  |
| --- |
| # Write a Python program to copy the contents of a file into another file, line by line.  # Function to copy contents from one file to another  def Copy(sourceFile, destinationFile):      try:          # Open the source file in read mode          with open(sourceFile, 'r') as src:              # Open the destination file in write mode              with open(destinationFile, 'w') as dest:                  # Read and write each line from source to destination                  for line in src:                      dest.write(line)          print(f"\nContents are copied from {sourceFile} to {destinationFile} Successfully.")      except FileNotFoundError:          print(f"\nThe file {sourceFile} does not exist.")      except IOError as e:          print(f"An error occurred: {e}")  # Take input from user:  sourceFile = input("\nEnter the source file name: ")  destinationFile = input("\nEnter the destination file name: ")  Copy(sourceFile, destinationFile) #Calling method |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.12. Use the OS module to perform :**  **1.12.1. Create a directory**  **1.12.2. Directory Listing**  **1.12.3. Search for “.py” files**  **1.12.4. Remove a particular file** |

**PROGRAM :**

|  |
| --- |
| #1.Use the OS module to perform      #1.1. Create a directory      #1.2. Directory Listing      #1.3. Search for “.py” files      #1.4. Remove a particular file  import os  def createDirectory(directory):      try:          os.makedirs(directory)          print(f"Directory '{directory}' created successfully.")      except FileExistsError:          print(f"Directory '{directory}' already exists.")      except Exception as e:          print(f"An error occurred: {e}")  def listDirectory(directory):      try:          files = os.listdir(directory)          print(f"Listing contents of '{directory}':")          for file in files:              print(file)      except FileNotFoundError:          print(f"Directory '{directory}' does not exist.")      except Exception as e:          print(f"An error occurred: {e}")  def searchPythonfiles(directory):      try:          py\_files = [f for f in os.listdir(directory) if f.endswith('.py')]          if py\_files:              print(f"'.py' files in '{directory}':")              for file in py\_files:                  print(file)          else:              print(f"No '.py' files found in '{directory}'.")      except FileNotFoundError:          print(f"Directory '{directory}' does not exist.")      except Exception as e:          print(f"An error occurred: {e}")  def removeFile(file\_path):      try:          os.remove(file\_path)          print(f"File '{file\_path}' removed successfully.")      except FileNotFoundError:          print(f"File '{file\_path}' does not exist.")      except Exception as e:          print(f"An error occurred: {e}")  while True:      print("\nOptions:")      print("1. Create a directory")      print("2. List directory contents")      print("3. Search for '.py' files")      print("4. Remove a particular file")      print("5. Exit")  choice = input("Enter your choice: ")      if choice == '1':          directory = input("\nEnter the directory name to create: ")          createDirectory(directory)      elif choice == '2':          directory = input("\nEnter the directory name to list: ")          listDirectory(directory)      elif choice == '3':          directory = input("\nEnter the directory name to search for '.py' files: ")          searchPythonfiles(directory)      elif choice == '4':          file\_path = input("\nEnter the file path to remove: ")          removeFile(file\_path)      elif choice == '5':          print("\nExiting the program.")          break      else:          print("Invalid choice. Please try again.") |

**OUTPUT:**

|  |
| --- |
|  |

|  |
| --- |
| **1.13. Create a simple banking application by using inheritance.** |

**PROGRAM :**

|  |
| --- |
| #Create a simple banking application by using inheritance.  class Account:      def \_\_init\_\_(self, owner, balance=0.0):          self.owner = owner          self.balance = balance      def deposit(self, amount):          if amount > 0:              self.balance += amount              print(f"Deposited {amount}. New balance is {self.balance}.")          else:              print("Deposit amount must be positive.")      def withdraw(self, amount):          if amount > 0:              if self.balance >= amount:                  self.balance -= amount                  print(f"Withdrew {amount}. New balance is {self.balance}.")              else:                  print("Insufficient funds.")          else:              print("Withdrawal amount must be positive.")      def display\_balance(self):          print(f"Account owner: {self.owner}, Balance: {self.balance}")  class SavingsAccount(Account):      def \_\_init\_\_(self, owner, balance=0.0, interest\_rate=0.01):          super().\_\_init\_\_(owner, balance)          self.interest\_rate = interest\_rate      def add\_interest(self):          interest = self.balance \* self.interest\_rate          self.balance += interest          print(f"Interest added. New balance is {self.balance}.")  class CurrentAccount(Account):      def \_\_init\_\_(self, owner, balance=0.0, overdraft\_limit=500):          super().\_\_init\_\_(owner, balance)          self.overdraft\_limit = overdraft\_limit      def withdraw(self, amount):          if amount > 0:              if self.balance + self.overdraft\_limit >= amount:                  self.balance -= amount                  print(f"Withdrew {amount}. New balance is {self.balance}.")              else:                  print("Overdraft limit exceeded.")          else:              print("Withdrawal amount must be positive.")  def main():  accounts = {}      while True:          print("\nBanking System Menu:")          print("1. Create Savings Account")          print("2. Create Current Account")          print("3. Deposit")          print("4. Withdraw")          print("5. Display Balance")          print("6. Add Interest (Savings Account only)")          print("7. Exit")            choice = input("\nEnter your choice: ")          if choice == '1':              owner = input("\nEnter account owner's name: ")              balance = float(input("\nEnter initial balance: "))              interest\_rate = float(input("\nEnter interest rate: "))              accounts[owner] = SavingsAccount(owner, balance, interest\_rate)              print("\nSavings account created successfully.")          elif choice == '2':              owner = input("\nEnter account owner's name: ")              balance = float(input("\nEnter initial balance: "))              overdraft\_limit = float(input("\nEnter overdraft limit: "))              accounts[owner] = CurrentAccount(owner, balance, overdraft\_limit)              print("\nCurrent account created successfully.")          elif choice == '3':              owner = input("\nEnter account owner's name: ")              if owner in accounts:                  amount = float(input("\nEnter amount to deposit: "))                  accounts[owner].deposit(amount)              else:                  print("\nAccount not found.")          elif choice == '4':              owner = input("\nEnter account owner's name: ")              if owner in accounts:                  amount = float(input("\nEnter amount to withdraw: "))                  accounts[owner].withdraw(amount)              else:                  print("\nAccount not found.")          elif choice == '5':              owner = input("\nEnter account owner's name: ")              if owner in accounts:                  accounts[owner].display\_balance()              else:                  print("\nAccount not found.")          elif choice == '6':              owner = input("\nEnter account owner's name: ")              if owner in accounts:                  account = accounts[owner]                  if isinstance(account, SavingsAccount):                      account.add\_interest()                  else:                      print("\nInterest can only be added to savings accounts.")              else:                  print("\nAccount not found.")          elif choice == '7':              print("\nExiting the program.")              break          else:              print("\nInvalid choice. Please try again.")  if \_\_name\_\_ == "\_\_main\_\_":      main() |

**OUTPUT:**

|  |
| --- |
| PS E:\MCA\Semesters\SEM 3\Python(MCA306)\Python\_LAB\Python\_LAB\_Cycle> & C:/Users/marsh/AppData/Local/Microsoft/WindowsApps/python3.11.exe "e:/MCA/Semesters/SEM 3/Python(MCA306)/Python\_LAB/Python\_LAB\_Cycle/p1\_13\_bankingApp.py"  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 1  Enter account owner's name: Anshad  Enter initial balance: 15000  Enter interest rate: 8  Savings account created successfully.  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 2  Enter account owner's name: Dilsha C P  Enter initial balance: 25000  Enter overdraft limit: 50000  Current account created successfully.  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 3  Enter account owner's name: Anshad  Enter amount to deposit: 20000  Deposited 20000.0. New balance is 35000.0.  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 4  Enter account owner's name: Dilsha C P  Enter amount to withdraw: 5000  Withdrew 5000.0. New balance is 20000.0.  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 5  Enter account owner's name: Anshad  Account owner: Anshad, Balance: 35000.0  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 6  Enter account owner's name: Dilsha C P  Interest can only be added to savings accounts.  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 6  Enter account owner's name: Anshad  Interest added. New balance is 315000.0.  Banking System Menu:  1. Create Savings Account  2. Create Current Account  3. Deposit  4. Withdraw  5. Display Balance  6. Add Interest (Savings Account only)  7. Exit  Enter your choice: 7  Exiting the program.  PS E:\MCA\Semesters\SEM 3\Python(MCA306)\Python\_LAB\Python\_LAB\_Cycle> |