Python Journal

Q1) Write a program to determine if a given string is palindrome or not using combination of positive and negative indexing. Take the string as an input from the user.

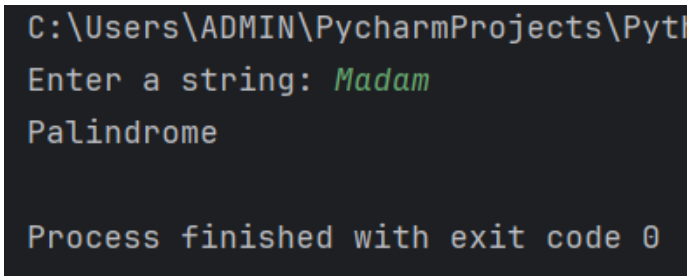
-> Check_Palindrome.py

```
class checkPalindrome:
    def __init__(self, s):
        self.s = s

    def palindromeCheck(self):  # method to check if the entered string is palindrome or not
        self.s = self.s.lower()  # converts the entered input into lowercase
        isPalindrome = True  # flag variable initially set to True
        length = len(self.s) // 2  # we take half the length of input since we need to match first half with second half
        for i in range(length):
            if self.s[i] != self.s[-(i + 1)]:
                # if the first and second half don't match, we make isPalindrome to False, thus indicating not a palindrome
                isPalindrome = False
        if isPalindrome:
            return "Palindrome"
        else:
            return "Not a palindrome"

string = input("Enter a string: ")   # takes input from the user
p = checkPalindrome(string)  # object creation
print(p.palindromeCheck())  # function call
```

Output:


C:\Users\ADMIN\PycharmProjects\Pyt
Enter a string: Madam
Palindrome
Process finished with exit code 0

Q2) Without using count() demonstrate the use of for loop to determine the number of occurences of a given character in a string. Take the string and character from the user.

-> Count_Occurences.py

```
class countOccurences:
    def __init__(self, s, ch):
        self.s = s
        self.ch = ch

    def occurencesOfChar(self):  # method to check the occurences of character in a string
        count = 0
        for i in self.s:
            if i == self.ch:  # if the occurence of character matches the index, we increment the count
                count += 1
        return count

s = input("Enter a string: ")   # takes input from the user
ch = input("Enter a character: ")
c = countOccurences(s, ch)  # object creation
print(c.occurencesOfChar())  # function call
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python_Jou
Enter a string: This is the tree of Teal
Enter a character: T
2
```

Q3) Without using readymade methods, write a program to find factorial of a given number. Take the number from the user.

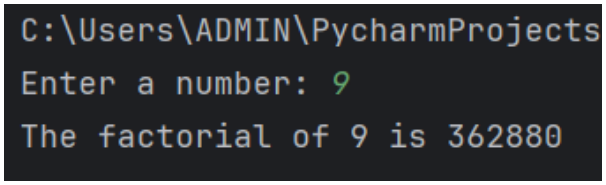
-> Calculate_Factorial.py

```
class calculateFact:
    def __init__(self, num):
        self.num = num


    def calculateFactorial(self):  # method to calculate the factorial of a number
        fact = 1
        for i in range(1, self.num + 1):
            fact = fact * i
        print(f"The factorial of {self.num} is {fact}")

n = int(input("Enter a number: "))   # takes input from the user
f = calculateFact(n)  # object creation
f.calculateFactorial()  # function call
```

Output:



Q4) Without using any readymade methods, write a program in Python to reverse the sequence of words in a given string. Take the string from the user.

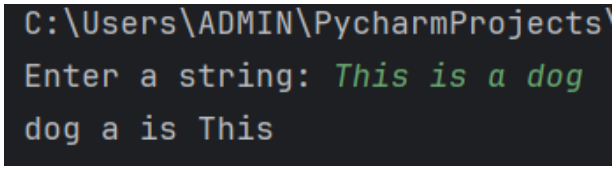
-> Reverse_Words.py

```
class reverseWords:
    def __init__(self, s):
        self.s = s

    def reverseWords(self):  # method to reverse the sequence of words in a string
        st = self.s.split()  # splits the words and creates a list
        reverse_str = st[::-1]  # reverse the order of words using indexing
        output = ' '.join(reverse_str)  # joins all the reverse words
        return output

s = input("Enter a string: ")   # takes input from the user
r = reverseWords(s)
print(r.reverseWords())  # function call
```

Output:



Q5) Without using any readymade methods, write a program in Python to check if the given number is an Armstrong number or not. Take the number from the user.

-> Check_Armstrong.py:

```
class checkArmstrong:
    def __init__(self, num):
        self.num = num

    def armstrongCheck(self):  # method to check if the number is armstrong or not
        temp = self.num  # temporary variable which holds same value as the entered input
        sum = 0
        while self.num > 0:
            rem = self.num % 10
            sum = sum + (rem * rem * rem)
            self.num = self.num // 10
        if sum == temp:
            print(f"{temp} is an Armstrong number")
        else:
            print(f"{temp} is not an Armstrong number")

n = int(input("Enter a number: "))   # takes input from the user
a = checkArmstrong(n)
a.armstrongCheck()  # function call
```

Output:

```
C:\Users\ADMIN\PycharmProjec
Enter a number: 153
153 is an Armstrong number
```

```
C:\Users\ADMIN\PycharmProjects
Enter a number: 25
25 is not an Armstrong number
```

Q6) Without using readline() demonstrate a way in Python to read a multiline file line by line.

-> Read_MultiLines.py # file containing all the file handling methods

```
class readMultipleLines:
    def __init__(self, file):
        self.file = file

    def readMultiLines(self):  # method to read a multiline file line by line without using readlines()
        try:
            f = open(self.file)  # opens the specified file in 'r' mode which is default mode
            f.seek(0)  # starts the file pointer from 0
            print(f"The contents of {self.file} are...")
            for i in f:
                print(i)  # prints the contents line by line
        except FileNotFoundError:
            print("File does not exist")

fname = input("Enter the file name: ")
r = readMultipleLines(fname)
r.readMultiLines()
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python_Practice
Enter the file name: file.txt
The contents of file.txt are...
This is an example of file handling in Python

This is the second line



This is the fourth line
```

Q7) Using readlines() demonstrate a way to return the total number of NON BLANK lines in a file.

-> Read_NonBlank_Lines.py

```
class readNonBlankLines:
    def __init__(self, file):
        self.file = file

    def nonBlankLines(self):  # method to return the total number of non-blank lines in a file
        try:
            f = open(self.file)
            lines = f.readlines()  # reads all the lines from the specified file
            count = 0
            for i in lines:
                if i.strip():  # removes extra whitespaces thus returning only non-blank lines
                    count += 1
            print(f"The number of non blank lines in {self.file} is {count}")
        except FileNotFoundError:
            print("File does not exist")

fname = input("Enter the file name: ")
n = readNonBlankLines(fname)
n.nonBlankLines()
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python_Practice\
Enter the file name: file.txt
The number of non blank lines in file.txt is 3
```

Q8) Using file writing methods, write a message from the user in a file. Show use of write when the file is in 'w' mode and 'a' mode.

-> Write_Message_In_File.py

```
class writeMessageInFile:
    def __init__(self, file, msg, ap_msg):
        self.file = file
        self.msg = msg
        self.ap_msg = ap_msg

    def writeReadDemo(self):  # method to demonstrate the usage of 'w' and 'a' mode in file
        try:
            f = open(self.file, 'w')  # opens the specified file in 'w' mode
            f.write(self.msg)  # writes the user defined message in the file
            print("Content written successfully...")
            f = open(self.file)
            print(f.read())  # prints the contents of file after writing
            print()
            f = open(self.file, 'a')  # opens the specified file in 'a' mode
            f.write(self.ap_msg)
            print("Content appended successfully...")
            f = open(self.file)
            print(f.read())
        except FileNotFoundError:
            print("File does not exist")

fname = input("Enter the file name: ")
message = input("Enter the message to write in file: ")
append_msg = input("Enter the message to append after writing: ")
rw = writeMessageInFile(fname, message, append_msg)  # object creation
rw.writeReadDemo()
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python_Practice\.venv\Scripts\python.exe C:\
Enter the file name: file1.txt
Enter the message to write in file: This is an example of 'w' mode
Enter the message to append after writing: This is an example of 'a' mode
Content written successfully...
This is an example of 'w' mode

Content appended successfully...
This is an example of 'w' mode This is an example of 'a' mode
```

Q9) Write a class Student having attributes, name, rollNumber, mathsMks, scienceMks and engMks. Use getters and setters for these attributes. Write another class Marksheet having the attributes totalMks and percentage. Define a method calculateMarks() and calculatePercentage(). Create a Student class object in Marksheet class. Assign name, roll number, maths, science and english marks to the student class object. Invoke calculateMarks() and calculatePercentage() using the data of this Student object.

-> Student.py

```
class Student:
    def __init__(self, name, rollNumber, mathsMks, scienceMks, engMks):
        self._name = name   # non-public attributes
        self._rollNumber = rollNumber
        self._mathsMks = mathsMks
        self._scienceMks = scienceMks
        self._engMks = engMks

    def setName(self, name):    # setter to set the value
        if len(name.strip()) == 0:
            print("Name field should not be empty.")
        else:
            self._name = name

    def getName(self):   # getter to get the value
        return self._name

    def setRollNumber(self, rollNumber):
        if len(rollNumber.strip()) == 0:
            print("Roll number should not be empty.")
        else:
            self._rollNumber = rollNumber

    def getRollNumber(self):
        return self._rollNumber
```

```
    def setMathsMks(self, mathsMks):
        if mathsMks < 0:
            print("Math marks should not be negative.")
        else:
            self._mathsMks = mathsMks

    def getMathsMks(self):
        return self._mathsMks

    def setScienceMks(self, scienceMks):
        if scienceMks < 0:
            print("Science marks should not be negative.")
        else:
            self._scienceMks = scienceMks

    def getScienceMks(self):
        return self._scienceMks

    def setEngMks(self, engMks):
        if engMks < 0:
            print("English marks should not be negative.")
        else:
            self._engMks = engMks

    def getEngMks(self):
        return self._engMks


-> from Student import Student
# import Student class from Student.py
class Marksheet:
    def __init__(self):
    self.totalMks = 0  # public attribute
```

```
        self.percentage = 0


    def calculateMarks(self, mathsMks, scienceMks, engMks):
        # method to calculate total marks
        self.totalMks = mathsMks + scienceMks + engMks
        print(f"The total marks are {self.totalMks}")

    def calculatePercentage(self):   # method to calculate total percentage
        self.percentage = (self.totalMks * 100) / 300
        print(f"The percentage of the student is
            {round(self.percentage, 2)}")

student = Student("", "", 0, 0, 0)
student.setName(input("Enter the student name: "))
student.setRollNumber(input("Enter the roll number of the student: "))
student.setMathsMks(int(input("Enter the maths marks: ")))
student.setScienceMks(int(input("Enter the science marks: ")))
student.setEngMks(int(input("Enter the english marks: ")))

marks = Marksheet()
marks.calculateMarks(student.getMathsMks(), student.getScienceMks(),
student.getEngMks())
marks.calculatePercentage()
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python_Practice
Enter the student name: Pratyush
Enter the roll number of the student: 2401121
Enter the maths marks: 90
Enter the science marks: 92
Enter the english marks: 91
The total marks are 273
The percentage of the student is 91.0
```

Q10) Using the concept of class, public and non-public attributes and methods write a program to calculate the area of a rectangle.

-> Calculate_Area_Rect.py

```
class calculateAreaRect:
    def __init__(self, length, width):
        self.length = length  # public attribute
        self.width = width
        self._area = 0.0  # non-public attribute

    def _calculateArea(self):  # non-public method
        self._area = self.length * self.width

    def getArea(self):  # public method
        self._calculateArea()  # call the _calcualateArea method
        return self._area  # returns the area of the rectangle

length = int(input("Enter the length of the rectangle: "))   # takes input from the user
width = int(input("Enter the width of the rectangle: "))
rec = calculateAreaRect(length, width)
print(f"The area of the rectangle is {rec.getArea()}")
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python
Enter the length of the rectangle: 5
Enter the width of the rectangle: 6
The area of the rectangle is 30
```

Q11) Write a class Employee having attributes name, dept, sal. Add a method calculate_salary(). This method should calculate the salary using the logic, 30* 2000. Print the final salary calculated. Write a subclass SalesEmployee having attribute no_of_leads. Override calculate_salary() which uses the formula, salary = 30*2000*no_of_leads. Write another subclass ManufacturingEmployee having attribute no_of_extra_hours. Override calculate_salary which uses the formula, salary = 30*20*no_of_extra_hours. In a separate file, EmployeeSalary.py create objects of these classes and invoke their respective calculate_salary().

->
```
class Employee:
    def __init__(self, name, dept, sal):
        self.name = name   # public attributes
        self.dept = dept
        self.sal = sal

    def calculate_salary(self):  # method to calculate salary
        total_sal = 30 * self.sal
        print(f"The total salary of the employee is {total_sal}")

class SalesEmployee(Employee):
    def __init__(self, name, dept, sal, no_of_leads):
        super().__init__(name, dept, sal)
        # call attributes of parent class using super()
        self.no_of_leads = no_of_leads

    def calculate_salary(self):   # override calculate_salary method
        total_sal = 30 * self.sal * self.no_of_leads
        print(f"The total salary of the sales employee is {total_sal}")

class ManufacturingEmployee(Employee):
    def __init__(self, name, dept, sal, no_of_extra_hours):
        super().__init__(name, dept, sal)
        self.no_of_extra_hours = no_of_extra_hours
```

```
    def calculate_salary(self):
        total_sal = 30 * self.sal * self.no_of_extra_hours
        print(f"The total salary of the manufacturing employee
            is  {total_sal}")
```

-> Employee_Salary.py

```
from Employee import Employee, SalesEmployee, ManufacturingEmployee
# Common user inputs
name = input("Enter the name of the employee: ")
dept = input("Enter the department of the employee: ")
salary = int(input("Enter the base salary of the employee: "))
# Dynamic user input based on type
employee_type = input("Enter the type of employee
(General/Sales/Manufacturing): ").lower()
if employee_type == "sales":   # for sales employee
    no_of_leads = int(input("Enter the number of leads generated
                                by the employee: "))
    sales_emp = SalesEmployee(name, dept, salary, no_of_leads)
    sales_emp.calculate_salary()
elif employee_type == "manufacturing":   # for manufacturing employee
    no_of_extra_hours = int(input("Enter the number of extra hours
                                    worked by the employee: "))
     manufacturing_emp = ManufacturingEmployee(name, dept, salary,
                                                    no_of_extra_hours)
    manufacturing_emp.calculate_salary()
else:   # for general employee
    emp = Employee(name, dept, salary)
    emp.calculate_salary()
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python_Practice\.venv\Scripts\pyt
Enter the name of the employee: Pratyush
Enter the department of the employee: IT
Enter the base salary of the employee: 6000
Enter the type of employee (General/Sales/Manufacturing): Sales
Enter the number of leads generated by the employee: 5
The total salary of the sales employee is 900000
```

Q12) Design a calculator utility module having methods for addition, subtraction, division and multiplication. Use this module in a different file which takes the number from the user and the choice of operation. Exhibit support for arbitrary arguments in addition and multiplication methods.

-> CalculatorUtility.py

```
def addition(*num):   # addition function with arbitrary argument
    res = 0
    for n in num:
        res += n
   print(f"The addition of numbers is {res}")

def subtraction(num1, num2):   # subtraction function
    if num1 > num2:
        print(f"The subtraction of numbers is {num1 - num2}")
    else:
        print(f"The subtraction of numbers is {num2 - num1}")

def division(num1, num2):   # division function
    if num1 > num2:
        print(f"The division of numbers is {num1 // num2}")
    else:
        print(f"The division of numbers is {num2 // num1}")

def multiplication(*num):   # multiplication function with arbitrary argument
    res = 1
    for n in num:
        res *= n
    print(f"The multiplication of numbers is {res}")
```

-> User_Input.py

```
import Calculatorutility as cal
num1 = int(input("Enter the first number: "))   # takes input from user
num2 = int(input("Enter the second number: "))
while True:
```

```
print("1) Addition")
print("2) Subtraction")
print("3) Multiplication")
print("4) Division")
print("5) Exit")
ch = int(input("Enter your choice: "))  # asks for user's choice
match ch:
        case 1:
                cal.addition(num1, num2)
        case 2:
                cal.subtraction(num1, num2)
        case 3:
                cal.multiplication(num1, num2)
        case 4:
                cal.division(num1, num2)
        case 5:
                break
        case _:
                print("Please enter a valid input")
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Pyth
Enter the first number: 5
Enter the second number: 5
1) Addition
2) Subtraction
3) Multiplication
4) Division
5) Exit
Enter your choice: 1
The addition of numbers is 10
1) Addition
2) Subtraction
3) Multiplication
4) Division
5) Exit
Enter your choice: 5

Process finished with exit code 0
```

Q13) You are developing an app for online ticket booking for an auditorium. The business allows per person to book maximum 5 tickets. If the number of tickets booked by a person goes beyond 5, the app should raise TicketsCountExceededError. Write a custom exception class for delivering this business requirement of the ticket booking app.

```
-> class TicketsCountExceededError(Exception):   # user defined exception
    def __init__(self, msg):
        super().__init__(msg)   # calls the msg from Exception class
        self.msg = msg

tcount = int(input("Enter the number of tickets you want to buy: "))
try:
        if tcount > 5:
        raise TicketsCountExceededError("You cannot buy more than 5 tickets..")
# call user defined exception
except TicketsCountExceededError as tc_error:   # tc_error is the alias name
        print(tc_error.msg)  # print the msg
else:
        print("Tickets booked successfully!!")
```

Output:

```
C:\Users\ADMIN\PycharmProjects\Python_Practice\
Enter the number of tickets you want to buy: 6
You cannot buy more than 5 tickets..
```

```
C:\Users\ADMIN\PycharmProjects\Python_Practice\
Enter the number of tickets you want to buy: 5
Tickets booked successfully!!
```