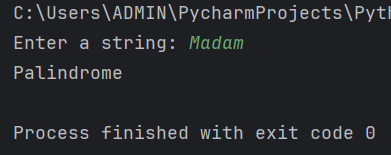
Python Journal

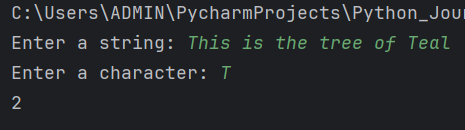
Q1) Write a program to determine if a given string is palindrome or not using combination of positive and negative indexing. Take the string as an input from the user.

-> 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:

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.

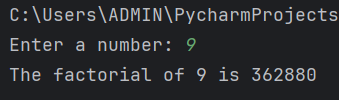
-> Count\_Occurences.py  
class countOccurences:  
 def \_\_init\_\_(self, s, ch):  
 self.s = s  
 self.ch = ch  
  
 def occurencesOfChar(self): # method to check the occurences of character in a string  
 count = 0  
 for i in self.s:  
 if i == self.ch: # if the occurence of character matches the 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:

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

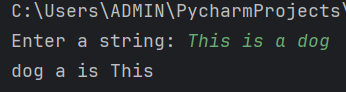
-> Calculate\_Factorial.py  
class calculateFact:  
 def \_\_init\_\_(self, num):  
 self.num = num

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:

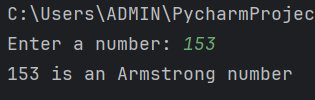
Q4) Without using any readymade methods, write a program in Python to reverse the sequence of words in a given string. Take the string from the user.

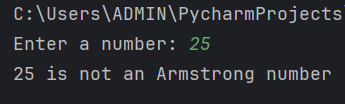
-> Reverse\_Words.py  
class reverseWords:  
 def \_\_init\_\_(self, s):  
 self.s = s  
  
 def reverseWords(self): # method to reverse the sequence of words in a string  
 st = self.s.split() # splits the words and creates a list  
 reverse\_str = st[::-1] # reverse the order of words using indexing  
 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.

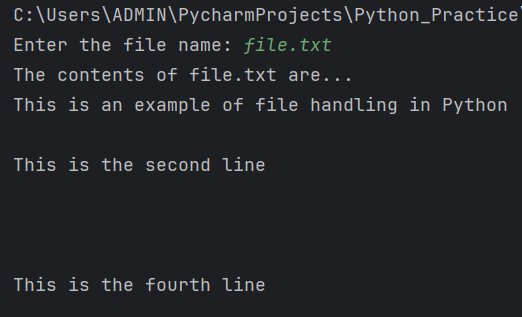
-> Check\_Armstrong.py:  
class checkArmstrong:  
 def \_\_init\_\_(self, num):  
 self.num = num  
  
 def armstrongCheck(self): # method to check if the number is armstrong or not  
 temp = self.num # temporary variable which holds same value as the entered input  
 sum = 0  
 while self.num > 0:  
 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:



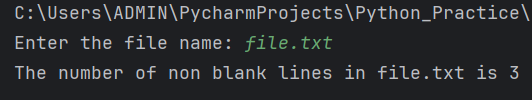
Q6) Without using readline() demonstrate a way in Python to read a multiline file line by line.

-> 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:

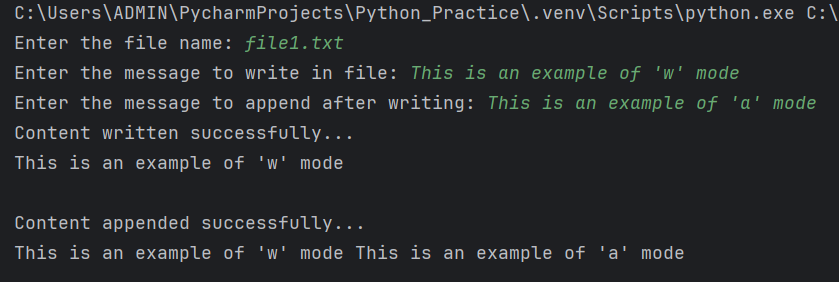
Q7) Using readlines() demonstrate a way to return the total number of NON BLANK lines in a file.

-> 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:

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()

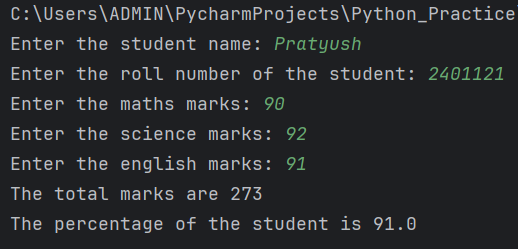
Output:

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

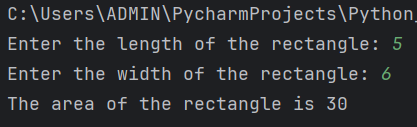
if len(name.strip()) == 0:  
 print("Name field should not be empty.")  
 else:  
 self.\_name = name  
  
def getName(self): # getter to get the value  
 return self.\_name  
  
def setRollNumber(self, rollNumber):  
 if len(rollNumber.strip()) == 0:  
 print("Roll number should not be empty.")  
 else:  
 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:

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

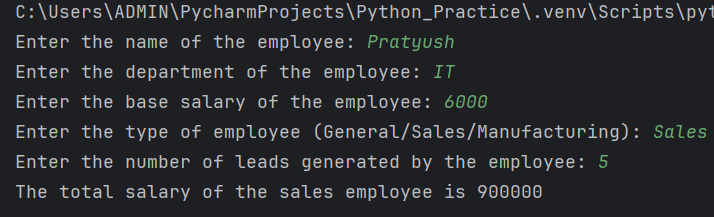
-> Calculate\_Area\_Rect.py  
class calculateAreaRect:  
 def \_\_init\_\_(self, length, width):  
 self.length = length # public attribute  
 self.width = width  
 self.\_area = 0.0 # non-public attribute  
  
 def \_calculateArea(self): # non-public method  
 self.\_area = self.length \* self.width  
  
 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:

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}")

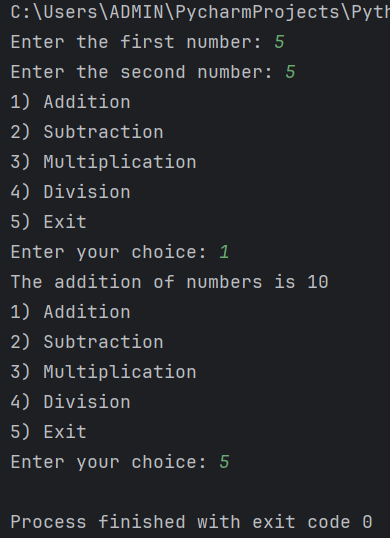
-> Employee\_Salary.py  
from Employee import Employee, SalesEmployee, ManufacturingEmployee  
# Common user inputs  
name = input("Enter the name of the employee: ")  
dept = input("Enter the department of the employee: ")  
salary = int(input("Enter the base salary of the employee: "))  
# Dynamic 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:

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.

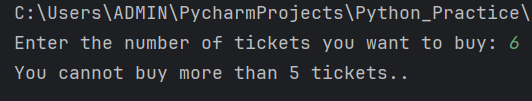
-> CalculatorUtility.py  
def addition(\*num): # addition function with arbitrary argument  
 res = 0  
 for n in num:  
 res += n  
 print(f"The addition of numbers is {res}")  
  
def subtraction(num1, num2): # subtraction function  
 if num1 > num2:  
 print(f"The subtraction of numbers is {num1 - 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:

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:

