labsheet1

June 3, 2024

[ ]:
```
Name: Sunil Paudel
Roll no.: PUR077BEI042
Subject: Artificial Intelligence
```

1. WAP to check if an input number is odd or even

[8]:
```
x = int(input("Enter a number: "))
def check(n):
    """
    check the number: odd or even

    Args:
    n (int): input

    Returns:
    prints the string
    """
    if (n % 2 == 0):
        print('Even Number')
    else:
        print('Odd Number')
check(x)
```

```
Enter a number:  10

Even Number
```

2. WAP to input the percentage and display the division >=80 → Distinction >=65 → First Division >=55 → Second Division >=40 → Third Division <40 → Fail

[37]:
```
x = int(input("Enter the percentage: "))

def division(x):
    """
    Gives Division of particular percentage

    Args:
    x (int) = input percentage

    returns:
```

```
    str: Division of percentage
    """
    if (x >= 80):
        return 'Distinction'
    elif (x >=65):
        return 'First Division'
    elif (x >= 55):
        return 'Second Division'
    elif (x >= 40):
        return 'Third Division'
    else:
        return 'Fail'

print(division(x))
```

```
Enter the percentage:  75

First Division
```

3. WAP to calculate sum, diff, product and quotient between two input numbers using a single function

[8]:
```
x = int(input("Enter 1st number: "))
y = int(input("Enter 2nd number: "))


def calculation(a, b):
    """
    performs basic arithmetic

    args:
    a, b (int): i/p's from user

    returns:
    prints values after arithmetic operations
    """
    print(f"Sum = {a+b}")
    print(f"Difference = { a - b}")
    print(f"Product = { a * b }")
    print(f"Quotient = { a / b}")
calculation(x, y)
```

```
Enter 1st number:  10
Enter 2nd number:  2

Sum = 12
difference = 8
Product = 20
Quotient = 5.0
```

4. WAP to display prime numbers from 1 to 100

[42]:
```
import math
x = int(input("Enter lower limit: "))
y = int (input("Enter Upper Limit: "))
list = []

def prime (x, y):
    """
    prints the prime numbers between given range

    args:
    x, y (int): lower & upper limit of range

    returns:
    list: a list of prime number
    """
    for i in range (x, y+1):
        count = 0
        sqrt = math.floor(math.sqrt(i))
        if (i != 1):
            for n in range(1, sqrt+1):
                if (i % n == 0):
                    count += 1
            if (count < 2):
                    list.append(i)
    print(list)

prime(x,y)
```

```
Enter lower limit:  1
Enter Upper Limit:  100
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97]
```

5. WAP to enter the marks of 10 students and display it.

[62]:
```
students = []
marks = []
for index in range (10):
    x = input(f"Enter the name of student {index}: ")
    students.append(x)
    y = int(input(f"Enter the marks of student {index}: "))
    marks.append(y)
for i in range (10):
    print(f"\'{students[i]}\' : {marks[i]}")
```

```
Enter the name of student 0:  y
Enter the marks of student 0:  79
Enter the name of student 1:  x
Enter the marks of student 1:  56
Enter the name of student 2:  z
Enter the marks of student 2:  49
Enter the name of student 3:  a
Enter the marks of student 3:  69
Enter the name of student 4:  q
Enter the marks of student 4:  58
Enter the name of student 5:  e
Enter the marks of student 5:  63
Enter the name of student 6:  u
Enter the marks of student 6:  38
Enter the name of student 7:  i
Enter the marks of student 7:  86
Enter the name of student 8:  p
Enter the marks of student 8:  43
Enter the name of student 9:  l
Enter the marks of student 9:  36
```

```
'y' : 79
'x' : 56
'z' : 49
'a' : 69
'q' : 58
'e' : 63
'u' : 38
'i' : 86
'p' : 43
'l' : 36
```

6. WAP to calculate the factorial of an input number.

[73]:
```
x = int(input("Enter a number: "))
def fact(n):
    """
    Calculates factorial of input number

    Args:
    n (int): i/p number

    Return:
    int: factorial of number
    """
    if (n == 1):
        return 1;
    else:
```

```
        return n * fact(n-1)
print(f"factorial of {x}: {fact(x)}")
```

```
Enter a number:  5

factorial of 5: 120
```

7. WAP to ask for a sentence and count the number of words.

[85]:
```
x = input("Enter a sentence: ")
def words_counter(x):
    count = 1
    for i in x:
        if (i == ' '):
            count += 1
    return count
print(f"Number of words: {words_counter(x)}")
```

```
Enter a sentence:  Let's Rick N' Roll

Number of words: 4
```

8. WAP to sort the list {5, 4, 11, 13, 51}

[90]:
```
def bubbleSort(list):
    """
    Sorts the element of list using BubbleSort

    Args:
    list (int): given list

    Return:
    list: a sorted list in ascending order
    """

    for i in range(len(list)):  # for traversal over list
        for j in range(0, len(list) - i - 1): # to compare list elements
          if list[j] > list[j + 1]: # check for bigger elements, followed by␣
 ↪swap if necessary
            temp = list[j]
            list[j] = list[j+1]
            list[j+1] = temp

data = [5, 4, 11, 13, 51]
bubbleSort(data)
print(f'Sorted list in Ascending Order: {data}')
```

```
Sorted list in Ascending Order: [4, 5, 11, 13, 51]
```

9. WAP program to sum all the items in a list.

[98]:
```
list = [1, 2, 3, 4, 5,10, 7]
def summ(li):
    """
    returns sum of all items in a list

    args:
    li (list) - a given list

    returns:
    int - sum of all elements in a list
    """
    sum = 0
    for i in range(len(li)):
        sum  += li[i]
    return sum
print(f"Total Sum: {summ(list)}")
```

```
Total Sum: 32
```

10. WAP program to get the largest number from a list.

[99]:
```
list = [ 5, 7, 10, 13, 25]
def largest(list):
    """
    gives largest element of the list

    args:
    list : given list

    returns:
    int : largest element of the list
    """
    large = list[0]
    for i in range(len(list)):
        if (list[i] > large):
            large = list[i]
    return large
print(f"Largest: {largest(list)}")
```

```
Largest: 25
```

11. WAP to ask for a sentence and calculate the frequency of characters in the sentences.

[38]:
```
x = input("Enter a sentence: ");
def char_counter(sentence):
    """
    Displays frequency of characters in the sentence
```

```
    args:
    sentence (string): i/p from user

    return:
    dict: a dict containing frequency of all characters in the given sentence
    """
    character_dict = {}
    for character in sentence:
        if character not in character_dict:
            character_dict[character] = 1
        else:
            character_dict[character] += 1
    return character_dict

print(char_counter(x))
```

```
Enter a sentence:  Real madrid are 15 times champions league winner!
```

```
{'R': 1, 'e': 6, 'a': 5, 'l': 2, ' ': 7, 'm': 3, 'd': 2, 'r': 3, 'i': 4, '1': 1,
'5': 1, 't': 1, 's': 2, 'c': 1, 'h': 1, 'p': 1, 'o': 1, 'n': 3, 'g': 1, 'u': 1,
'w': 1, '!': 1}
```

12. WAP to find the sum of all items in a dictionary.

[9]:
```
x = {'a':100, 'b':200, 'c':300}
def summ(li):
    """
    returns sum of all items in a list

    args:
    li (list) - a given list

    returns:
    int - sum of all elements in a list
    """
    sum = 0
    for i in range(len(li)):
        sum  += li[i]
    return sum
def sum_values(val):
    """
    returns sum of values of a dictionary

    args:
    val: given dictionary

    returns:
```

```
    int: sum of all values in dictionary
    """
    sum = 0
    values_list = list(val.values())
    return summ(values_list)

print(f"Sum of Values:{sum_values(x): }")
```

```
Sum of Values: 600
```

13. You are given a string and your task is to swap cases. In other words, convert all lowercase letters to uppercase letters and vice versa.

[17]:
```
x = input("Enter a String: ")
def swap(letters):
    """
    swaps lowercase to uppercase letter and vice-versa.

    args:
    letters: characters in given string

    returns:
    string: string after swapping cases of letters
    """
    output = ''
    for let in letters:
        if let.isupper():
            output += let.lower()
        elif let.islower():
            output += let.upper()
    return output
print(f"Before Swapping Cases: {x}, After Swapping: {swap(x)}")
```

```
Enter a String:  One Love
```

```
Before Swapping Cases: One Love, After Swapping: oNElOVE
```

14. Write a Python program to create a class representing a Circle. Include methods to calculate its area and perimeter.

[42]:
```
import math
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        """
        returns area of circle
        """
        return math.pi * self.radius**2
```

```
    def perimeter(self):
        """
        returns perimeter of circle
        """
        return 2 * math.pi * self.radius
x = float(input("Enter radius of circle"))
c = Circle(x)
print(f"Area of Cirlce with {x} radius: {c.area()}")
print(f"Perimeter of Cirlce with {x} radius: {c.perimeter()}")
```

```
Enter radius of circle 5

Area of Cirlce with 5.0 radius: 78.53981633974483
Perimeter of Cirlce with 5.0 radius: 31.41592653589793
```

15. Write a Python program to create a person class. Include attributes like name, country and date of birth. Implement a method to determine the person's age.

[9]:
```
from datetime import datetime
class person():
    def __init__(self, name, country, dob):
        self.name = name
        self.country = country
        self.dob = dob
    def calc(self):
        return datetime.now().year - self.dob
xyz = person ("Khosta", "Nepal", 2002)
print(xyz.calc())
```

```
22
```

16. Define a class Vehicle with attributes make and model, and a method drive() which prints "Driving the [make] [model]". Then, create a subclass Car that inherits from Vehicle and overrides the drive() method to print "Driving the [make] [model] car".

[14]:
```
class vehicle():
    def __init__(self, make, model):
        self.make = make
        self.model = model
    def drive(self):
        return f"Driving the {self.make} {self.model}"

class car(vehicle):
    def drive(self):
        return f"Driving the {self.make} {self.model}"

abc = vehicle ("Porsche","Boxter")
print(abc.drive())
print("overriding...")
abc = car ("BYD", "Dolphin")
```

```
print(abc.drive())
```

```
Driving the Porsche Boxter
overriding…
Driving the BYD Dolphin
```

17. Create a class BankAccount with private attributes balance and account_number. Implement methods deposit() and withdraw() to modify the balance. Ensure that the balance cannot be accessed directly from outside the class.

[16]:
```
class BankAccount():
    def __init__(self, balance, account_num):
        self.__balance = balance
        self.__account_num = account_num

    def deposit(self, balance):
        self.__balance += balance
        return self.__balance
    def withdraw(self, balance):
        if (balance > self.__balance ):
            return "Insufficient balance"
        self.__balance -= balance
        return self.__balance

account = BankAccount(25000, 98258)
print(f"The balance after depositing Rs. 7500 is Rs. {account.deposit(7500)}")
print(f"The balance after withdrawl of Rs. 10000 is Rs. {account.
 ↪withdraw(10000)}")
print(f"Trying to withdraw Rs. 23000: {account.withdraw(23000)}")
```

```
The balance after depositing Rs. 7500 is Rs. 32500
The balance after withdrawl of Rs. 10000 is Rs. 22500
Trying to withdraw Rs. 23000: Insufficient balance
```

18. Implement a class Shape with a method area() which returns 0. Then, create subclasses Rectangle and Circle. Overload the area() method in both subclasses to calculate and return the area of a rectangle and a circle respectively.

[19]:
```
import math
class Shape():
    def area(self):
        return 0
class Rectangle(Shape):
    def __init__(self, length, breadth):
        self.length = length
        self.breadth = breadth
    def area(self):
        return self.length * self. breadth
```

```
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area (self):
        return math.pi * self.radius ** 2

rectangle = Rectangle (3,2)
print(f"The area of rectangle is {rectangle.area()}")
circle = Circle(7)
print(f"The area of circle is {circle.area()}")
```

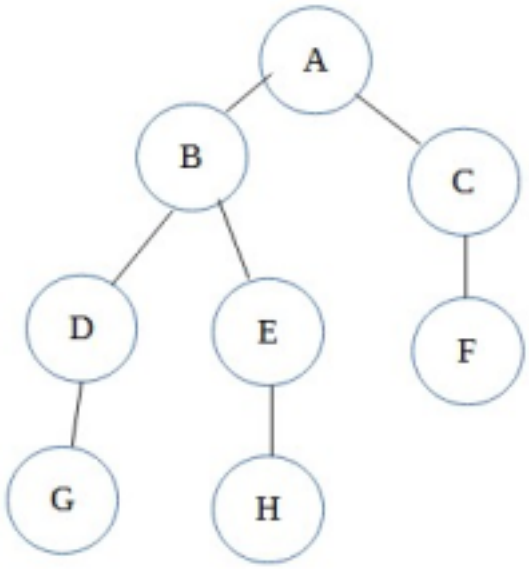
```
The area of rectangle is 6
The area of circle is 153.93804002589985
```

19. Define classes Engine, Wheel, and Car. Engine and Wheel classes have attributes type and methods start() and stop(). The Car class should have instances of Engine and Wheel classes as attributes. Implement a method start_car() in the Car class which starts the engine and prints "Car started".

[29]:
```
class Engine():
    def __init__(self, type):
        self.type = type
    def start(self):
        return f"Engine {self.type} Started"
    def stop(self):
        return f"Engine {self.type} Stopped"

class Wheel:
    def __init__(self, type):
        self.type = type
    def start(self):
        return f"Wheel {self.type} started rotating"
    def stop(self):
        return f"Wheel {self.type} stopped rotating"

class Car():
    def __init__(self, engine, wheel):
        self.engine = engine
        self.wheel = wheel
    def start_car(self):
        print(self.engine.start())
        print("Car Started")

engine1 = Engine("V7")
wheel1 = Wheel("Titanium")
car1 = Car(engine1, wheel1)
car1.start_car()
```

```
Engine V7 Started
Car Started
```

20. WAP to represent the following graphs using a dictionary.

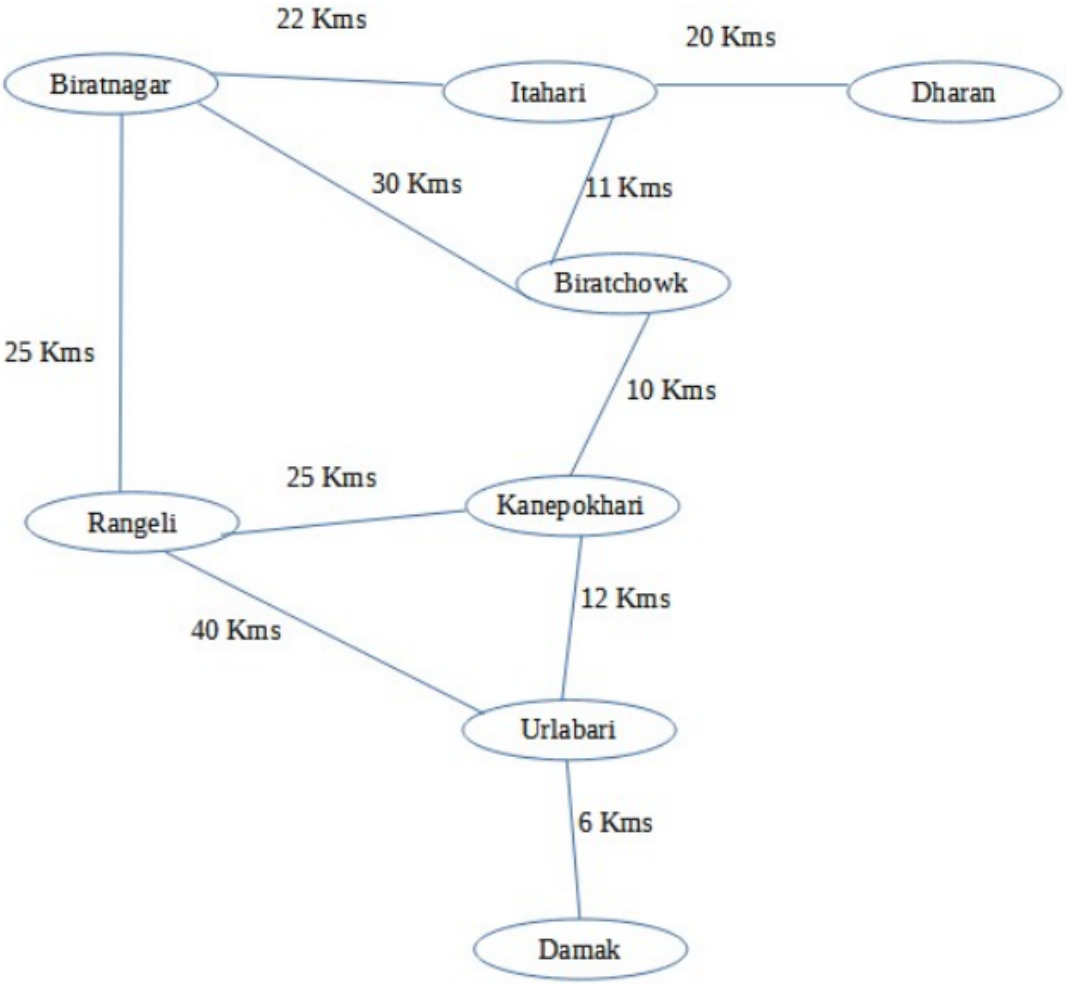


a.

[30]:
```
graphs_collector_dict = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B', 'G'],
    'E': ['B', 'H'],
    'F': ['C'],
    'G': ['D'],
    'H': ['E']
}
graph_final = dict()

def graph_creator():
    no_of_vertices = int(input("Enter the number of vertices: "))
    for i in range(no_of_vertices):
        vertex = input("Enter the vertex: ")
        edges = input("Enter the all Neighbour Vertices separated by comma: ").
 ↪split(",")
        graph_final[vertex] = edges
    return graph_final
```

```
print(f"The graph created is: {graph_creator()}")
```

```
Enter the number of vertices:  8
Enter the vertex:  A
Enter the all Neighbour Vertices separated by comma:  B, C
Enter the vertex:  B
Enter the all Neighbour Vertices separated by comma:  A, E, D
Enter the vertex:  C
Enter the all Neighbour Vertices separated by comma:  A, F
Enter the vertex:  D
Enter the all Neighbour Vertices separated by comma:  B, G
Enter the vertex:  E
Enter the all Neighbour Vertices separated by comma:  B,H
Enter the vertex:  F
Enter the all Neighbour Vertices separated by comma:  C
Enter the vertex:  G
Enter the all Neighbour Vertices separated by comma:  D
Enter the vertex:  H
Enter the all Neighbour Vertices separated by comma:  E
The graph created is: {'A': ['B', ' C'], 'B': ['A', ' E', ' D'], 'C': ['A', '
F'], 'D': ['B', ' G'], 'E': ['B', 'H'], 'F': ['C'], 'G': ['D'], 'H': ['E']}
```



b.

[1]:
```
class WeightedGraphCreator():
    def __init__(self):
        self.graph = dict()

    def add_edge(self, src, dest, weight):
        if src in self.graph:
            self.graph[src].append((dest, weight))
        else:
            self.graph[src] = [(dest, weight)]
        return self.graph
    def output_graph(self):
        return self.graph
```

```python
g = WeightedGraphCreator()
g.add_edge("Biratnagar", "Itahari", 22)
g.add_edge("Itahari", "Dharan", 20)
g.add_edge("Biratnagar", "Biratchowk", 30)
g.add_edge("Itahari", "Biratchowk", 11)
g.add_edge("Biratnagar", "Rangeli", 25)
g.add_edge("Rangeli", "Kanepokhari", 25)
g.add_edge("Biratchowk", "Kanepokhari", 10)
g.add_edge("Kanepokhari", "Urlabari", 12)
g.add_edge("Rangeli", "Urlabari", 40)
g.add_edge("Urlabari", "Damak", 6)
g.add_edge("Itahari", "Biratnagar", 22)
g.add_edge("Dharan", "Itahari", 20)
g.add_edge("Biratchowk", "Itahari", 11)
g.add_edge("Biratchowk", "Biratnagar", 30)
g.add_edge("Rangeli", "Biratnagar", 25)
g.add_edge("Kanepokhari", "Rangeli", 25)
g.add_edge("Kanepokhari", "Biratchowk", 10)
g.add_edge("Urlabari", "Kanepokhari", 12)
g.add_edge("Urlabari", "Rangeli", 40)
g.add_edge("Damak", "Urlabari", 6)


print(f"The weighted graph is: {g.output_graph()}")
```

```
The weighted graph is: {'Biratnagar': [('Itahari', 22), ('Biratchowk', 30),
('Rangeli', 25)], 'Itahari': [('Dharan', 20), ('Biratchowk', 11), ('Biratnagar',
22)], 'Rangeli': [('Kanepokhari', 25), ('Urlabari', 40), ('Biratnagar', 25)],
'Biratchowk': [('Kanepokhari', 10), ('Itahari', 11), ('Biratnagar', 30)],
'Kanepokhari': [('Urlabari', 12), ('Rangeli', 25), ('Biratchowk', 10)],
'Urlabari': [('Damak', 6), ('Kanepokhari', 12), ('Rangeli', 40)], 'Dharan':
[('Itahari', 20)], 'Damak': [('Urlabari', 6)]}
```