Lab 1: Introduction to Python

Done By:

- Name: Sudip Parajuli
- Roll: PUR077BEI041
- Faculty: BEI

Submitted To:

- Name: Pukar Karki
- Department of Electronics and Computer Engineering
- IOE Purwanchal Campus, Dharan

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

In [ ]:
```
number = input("Please Enter a Number to Check if it is Odd or Even: ")
number = int(number)

def odd_or_even(number: int) -> bool:
    """
    This function checks if a number is odd or even
    :param number: int
    :return: bool

    """
    if number % 2 ==0:
        return True
    else:
        return False


print(odd_or_even(number))
```

```
False
```

2)WAP to input the percentage and display the division

> =80 → Distinction =65 → First Division =55 → Second Division =40 → Third Division <40 → Fail

In [ ]:
```
percentage = input("Enter the Percentage you want to convert to division (0-

percentage_float = float(percentage)

def percentage_to_division_conversion(number: float) -> str:
    """
    Convert the Input Percentage into Division
```

```
    Args:
    number: float

    Returns:
    str

    """
    if number > 100 or number < 0:
        return "Invalid Percentage! Please Enter a Percentage between 0 and
    if number >= 80:
        return "Distinction"
    elif number >= 60:
        return "First Division"
    elif number >= 50:
        return "Second Division"
    elif number >= 40:
        return "Third Division"
    else:
        return "Fail"


print(percentage_to_division_conversion(percentage_float))
```

```
Third Division
```

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

In [ ]:
```
a = float(input("Enter the First Number: "))
b = float(input("Enter the Second Number: "))


def sum_diff_quotient_finder(a:float, b:float) -> tuple:
    """
    Finds the sum, difference and quotient

    Args:
        a: float
        b: float
    Returns:
        tuple
    """
    sum = a+b
    diff = abs(a-b)
    quotient = a/b

    output = (sum, diff, quotient)

    return output


print(f'The Sum, Difference and Quotient is {sum_diff_quotient_finder(a, b)}
```

```
The Sum, Difference and Quotient is (19.5, 4.5, 1.6) respectively
```

4)WAP to display prime numbers from 1 to 100

In [ ]:
```
import math
def prime_btn_one_to_hundred()-> list:
    """
        Calculates the Prime Number Between 1 to 100 using Sieve of Eratosth

        Args:
         None
        Returns:
         list: Returns a list containing all the prime numbers from 1 to 100

    """
    n=50
    A = [True] * (n+1)
    A[0] = A[1] = False

    for i in range(2, int(math.sqrt(n))+1):
        if A[i]:
            j = i*i
            while j <= n:
                A[j] = False
                j+=i
    return [i for i in range(2, n+1) if A[i]]

print(prime_btn_one_to_hundred())
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

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

In [ ]:
```
def enter_marks_of_10_students():
    """

    This function takes the name and marks of 10 students and returns a dict
    Args:
    None

    Returns:
    dict: A dictionary with the name as the key and marks as the value.

    """
    display = {}
    for i in range(10):
        name = input(f"Enter name of student {i+1}: ")
        mark = input (f"Enter marks of student {i+1}: ")
        display[name] = mark

    return display

print(f'The name and marks of the students are {enter_marks_of_10_students()}
```

```
The name and marks of the students are {'RAm': '12', 'Shyam': '10', 'Hari':
'98', 'Gita': '89', 'Sita': '23', 'Sudip': '87', 'Suwash': '98', 'Pawan': '7
9', 'Dhiraj': '67', 'Pratik': '84'}
```

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

In [ ]:
```
num = int(input("Enter a number: "))

def factorial_finder(num:int)-> int:
    """
    This function takes a number as input and returns the factorial of that
    Args:
    num(int): The number whose factorial is to be found.

    Returns:
    int: The factorial of the number.
    """
    if num == 0:
        return 1
    else:
        return num * factorial_finder(num-1)

print(f'The factorial of {num} is {factorial_finder(num)}')
```

```
The factorial of 78 is 113242811782062978314575211587320462287317495794882519900489628256688353252342007662450862131773440000000000000000000
```

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

In [ ]:
```
sentence = input("Enter a sentence: ")

def words_frequency(sentence:str)->dict:
    """
    This function takes a sentence as input and returns a dictionary of word
    Args:
        sentence (str): sentence to be processed
    Returns:
        dict: dictionary of words and their frequency

    """
    words_dict = dict()
    for word in sentence:
        if word in words_dict:
            words_dict[word] += 1
        else:
            words_dict[word] = 1
    return words_dict


print (f'The Words and corresponding frequency is {words_frequency(sentence)}')
```

```
The Words and corresponding frequency is {'T': 1, 'h': 1, 'e': 1, ' ': 4, 'Q': 1, 'u': 2, 'i': 1, 'c': 1, 'k': 1, 'B': 1, 'r': 1, 'o': 2, 'w': 1, 'n': 1, 'F': 1, 'x': 1, 'J': 1, 'm': 1, 'p': 1, 's': 1}
```

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

In [ ]:
```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
```

```
                arr[j], arr[j + 1] = arr[j + 1], arr[j]

arr = [5, 4, 11, 13, 51]
bubble_sort(arr)
print(arr)
```

```
[4, 5, 11, 13, 51]
```

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

In [ ]:
```
items = int(input("Enter the total no of items you want to sum: "))
sum = 0
for i in range(0, items):
    sum += int(input("Enter the number: "))
print("The sum of the numbers is: ", sum)
```

```
The sum of the numbers is:  4
```

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

In [ ]:
```
lists = [12, 23, 34, 45, 56, 67, 78, 89, 90]

def largest_element_in_the_list(lists):
    largest = lists[0]
    for i in range(1, len(lists)):
        if lists[i] > largest:
            largest = lists[i]
    return largest

print("The largest element in the list is: ", largest_element_in_the_list(li
```

```
The largest element in the list is:  90
```

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

In [ ]:
```
#words frequency calculator
string_data = input("Enter the string data: ")
frequency_dict = dict()

def sentence_frequency(string_data):
    for char in string_data:
        if char in frequency_dict:
            frequency_dict[char] += 1
        else:
            frequency_dict[char] = 1
    return frequency_dict


print(f"The frequency of the characters in the string is: {sentence_frequenc
```

```
The frequency of the characters in the string is: {'h': 1, 'e': 1, 'l': 2, '
o': 1}
```

12)WAP to find the sum of all items in a dictionary Input: {'a': 100, 'b':200, 'c':300} Output:

600

In [ ]:
```
dict_data = { 'a': 100, 'b': 200, 'c': 300}

def sum_of_values_dict(dict_data):
    sum = 0
    for value in dict_data.values():
        sum += value
    return sum

print(f"The sum of the values in the dictionary is: {sum_of_values_dict(dict
```

```
The sum of the values in the dictionary is: 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.

In [ ]:
```
string_data = input("Enter the string data: ")

def string_lowercase_uppercase_swapper(string_data):
    """
    This function swaps the case of the string data
    Initially it checks if the character is in lowercase or uppercase range

    Args:
    string_data: The string data to be swapped

    Returns:
    swapped_string: The string data after swapping the case


    """
    swapped_string = ""
    for char in string_data:
        if char >= 'a' and char <= 'z':
            swapped_string += char.upper()
        elif char >= 'A' and char <= 'Z':
            swapped_string += char.lower()
        else:
            swapped_string += char
    return swapped_string


print(f"The string after swapping the case is: {string_lowercase_uppercase_s
```

```
The string after swapping the case is: THe UPPERcASE
```

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

In [ ]:
```
class Circle():
    def __init__(self, radius):
        self.radius = radius

    def area_of_circle(self):
        return 3.14 * self.radius * self.radius
```

```
    def perimeter_of_circle(self):
        return 2 * 3.14 * self.radius


circle1 = Circle(5)
print(f"The area of the circle is: {circle1.area_of_circle()}")
print(f"The perimeter of the circle is: {circle1.perimeter_of_circle()}")
```

```
The area of the circle is: 78.5
The perimeter of the circle is: 31.400000000000002
```

Artifical Intelligence

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.

In [ ]:
```
from datetime import datetime
class Person():
    def __init__(self, name, country, dob):
        self.name = name
        self.country = country
        self.dob = dob

    def age_of_person(self):
        return datetime.now().year - self.dob


ram = Person("Ram", "Nepal", 1990)
print(f"The age of the person is: {ram.age_of_person()}")
```

```
The age of the person is: 34
```

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".

In [ ]:
```
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} car"


lamborghini = Vehicle("Lamborghini", "Aventador")
```

```
print(lamborghini.drive())
lamborghini = Car("Lamborghini", "Aventador")
print(lamborghini.drive())
```

```
Driving the Lamborghini Aventador
Driving the Lamborghini Aventador car
```

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.

In [ ]:
```
class BankAccount():
    def __init__(self, balance, account_number):
        self.__balance = balance
        self.__account_number = account_number

    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

account1 = BankAccount(1000, 123456)
deposit_amnt = account1.deposit(500)
print(f"The balance after depositing is {deposit_amnt}")
withdraw_amnt = account1.withdraw(500)
print(f"The balance after withdrawing is {withdraw_amnt}")
```

```
The balance after depositing is 1500
The balance after withdrawing is 1000
```

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.

In [ ]:
```
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 3.14 * self.radius * self.radius

rectangle1 = Rectangle(5, 10)
print(f"The area of the rectangle is: {rectangle1.area()}")
circle1 = Circle(5)
print(f"The area of the circle is: {circle1.area()}")
```

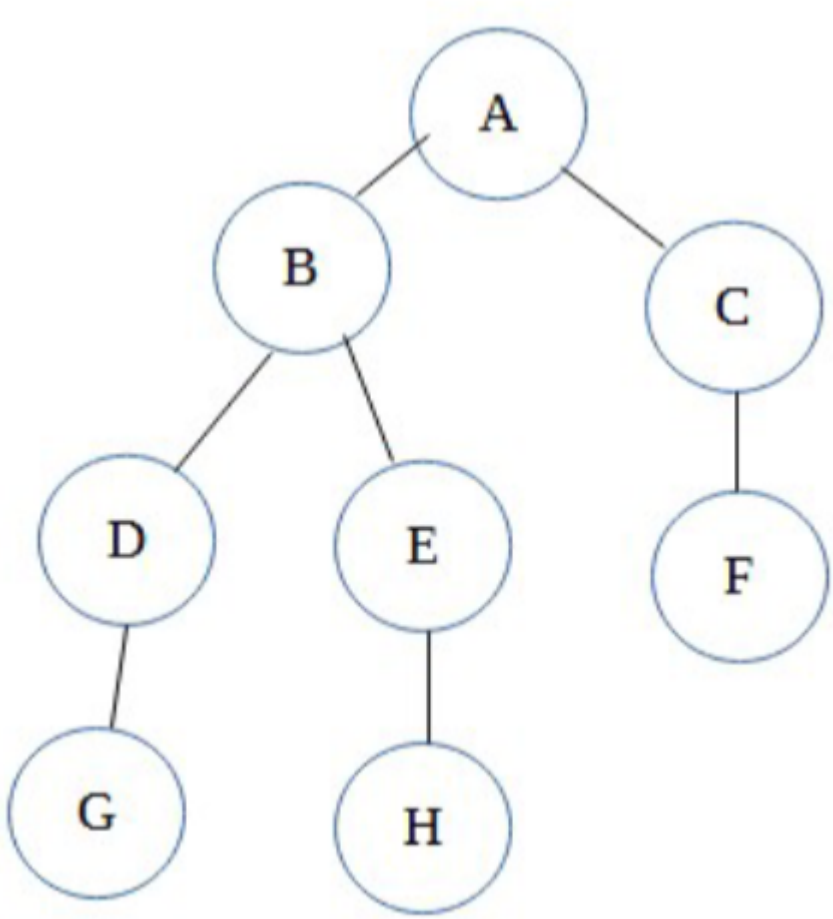
```
The area of the rectangle is: 50
The area of the circle is: 78.5
```

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".

In [ ]:
```
class Engine():
    def __init__(self, type):
        self.type = type
    def start(self):
        return f"Engine {self.type} started"
    def stop(self):
        return "Engine {self.type} stopped"


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

class Car():
    #instances of engine and wheel as attributes
    def __init__(self, engine, wheel):
        self.engine = engine
        self.wheel = wheel
    def start_car(self):
        print(self.engine.start())
        print("Car Started")


engine1 = Engine("V8")
wheel1 = Wheel("Alloy")
car1 = Car(engine1, wheel1)
car1.start_car()
```

```
Engine V8 started
Car Started
```

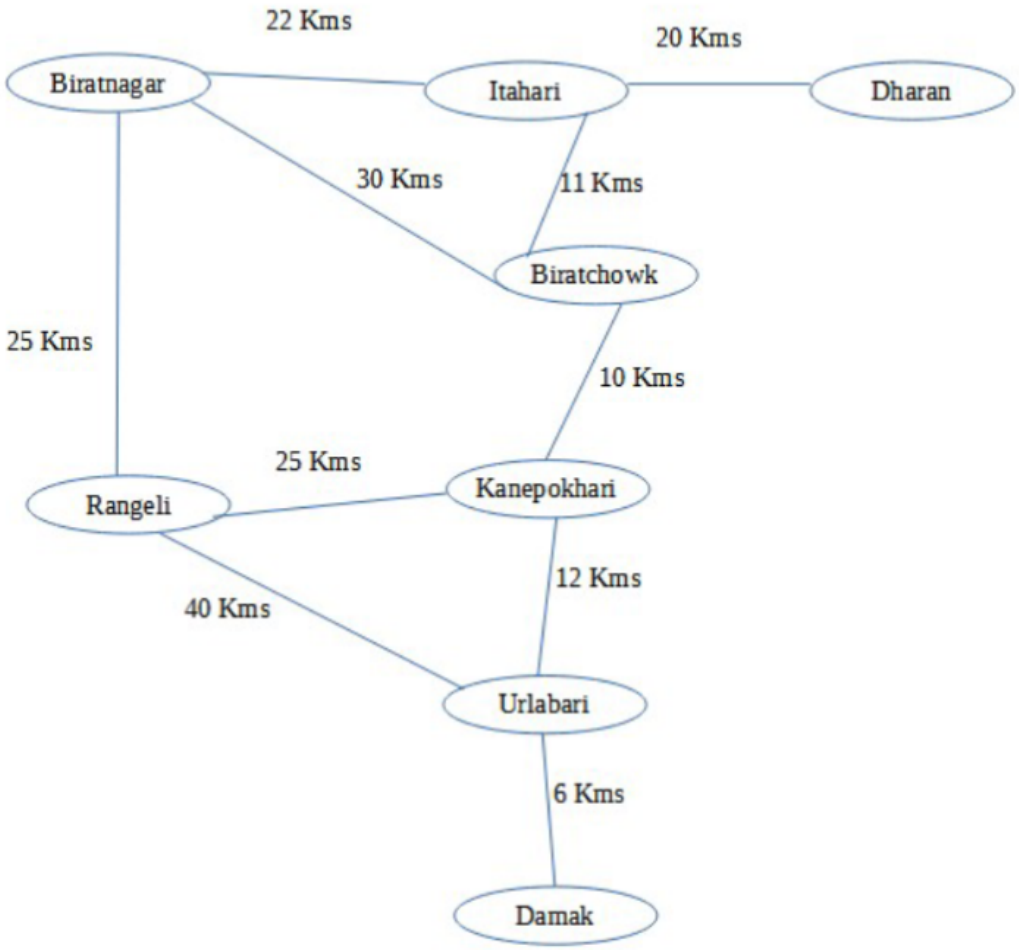
20. WAP to represent the following graphs using a dictionary. a)



In [ ]:
```
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:
        graph_final[vertex] = edges
    return graph_final

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

```
The graph created is: {'A': ['B', 'C'], 'B': ['A', 'D', 'E'], 'C': ['A',
'F'], 'D': ['B', 'G'], 'E': ['B', 'H'], 'F': ['C'], 'G': ['D'], 'H': ['E']}
```

b)



In [ ]:
```python
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


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), ('Biratnag
ar', 22)], 'Rangeli': [('Kanepokhari', 25), ('Urlabari', 40), ('Biratnagar',
25)], 'Biratchowk': [('Kanepokhari', 10), ('Itahari', 11), ('Biratnagar', 3
0)], 'Kanepokhari': [('Urlabari', 12), ('Rangeli', 25), ('Biratchowk', 10)],
'Urlabari': [('Damak', 6), ('Kanepokhari', 12), ('Rangeli', 40)], 'Dharan':
[('Itahari', 20)], 'Damak': [('Urlabari', 6)]}
```