DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

AI & Decision Support Systems

Lab Report #1

Student Name: **Nawab Aarij Imam**

Degree/ Syndicate: **43 CE - A**

Task1:

Q1: Write a program that lets the user enter in some English text, then converts the text to Pig-Latin.

To review, Pig-Latin takes the first letter of a word, puts it at the end, and appends “ay”. The only exception is if the first letter is a vowel, in which case we keep it as it is and append “hay” to the end.

E.g. “hello” -> “ellohay”, and “image” -> “imagehay”

Hint: *Split the entered string through split() method and then iterate over the resultant list, e.g. “My name is John Smith”.split(“ ”) -> [“My”, “name”, “is”, “John”, “Smith”]*

Code:

```
import numpy as np


def pig_latin(text):
    # check whitespace
    vowels = ['A','E','I','O','U','a','e','i','o','u']
    word_array = text.split(" ")
    for i, word in enumerate(word_array):
        if word[0] in vowels:
            word_array[i] = word + 'hay'
        else:
            word_array[i] = word[1:] + word[0] + 'ay'

    return ' '.join(word_array)


if __name__ == '__main__':
    input_string = input("Enter a string: ")
    print(pig_latin(input_string))
```

Output:

```
Enter a string: EME Core Hai
EMEhay oreCay aiHay
```

Task2:

Q2: Write a method to calculate Fibonacci series up to 'n' points. After calculating the series, the method should return it to main.

Code:

```
import numpy as np


def fibonacci(n):
    series = [0,1]
    for i in range(n)[1:]:
        series.append(series[i-1] + series[i])
    return series



if __name__ == '__main__':
    print(fibonacci(10))
```

Output:

```
ces&Technology/NUST MATERIAL/7th Semester
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

Task3:

Q3: Write a simple program that builds a random password generator. For password generator the user must enter total number of passwords and their lengths. Display all the passwords with random characters.

Code:

```
import random
import string


def generate_password(length):
    characters = string.ascii_letters + string.punctuation + string.digits
    password = ''.join(random.choice(characters) for i in range(length))
    return password


if __name__ == '__main__':
    total_passwords = int(input('Enter number of passwords: '))
    length = int(input('Enter length of passwords: '))
    passwords = []
    for i in range(total_passwords):
```

```
    passwords.append(generate_password(length))

  print(f'Your passwords are: ')
  for i, pw in enumerate(passwords):
    print(f'  {i+1}. {pw}')
```

Output:

```
Enter number of passwords: 5
Enter length of passwords: 8
Your passwords are:
  1. t03rv1X*
  2. *gVq?RiW
  3. #xxOWT0s
  4. M9a;_1J^
  5. IGKy\%P3
```

Task4:

Q4: Create a class named 'Complex' that must have the following attributes:
Variables named 'Real' and 'Imaginary'
Methods named Magnitude () and Orientation ()
Take a complex number from user in main and print its magnitude and orientation. You have a liberty to create methods signature as you like.

Code:

```
import math

class Complex:
  def __init__(self,real,img):
    self.real = real
    self.img = img

  def mag(self):
    return math.sqrt(self.real ** 2 + self.img ** 2)

  def orient(self):
    return math.atan(self.img / self.real)
```

```python
if __name__ == '__main__':
    num1 = Complex(3,5)
    print(num1.mag())
    print(num1.orient())
```

Output:

```
2.23606797749979
1.1071487177940906
```

Task5:

Q5: Create the following Binary Search Tree and search for the node ‘13’. You can hard code the tree as well, but it is better if you create it dynamically at run time (You must have learned in Data Structures & Algorithms). Also, tell the time performance of searching the node ‘13’ in Big-O notation.

Code:

```python
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.value = key


class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self._insert(self.root, key)

    def _insert(self, current_node, key):
```

```python
        if key < current_node.value:
            if current_node.left is None:
                current_node.left = Node(key)
            else:
                self._insert(current_node.left, key)
        elif key > current_node.value:
            if current_node.right is None:
                current_node.right = Node(key)
            else:
                self._insert(current_node.right, key)

    def search(self, key):
        return self._search(self.root, key)

    def _search(self, current_node, key):
        if current_node is None:
            return False
        if key == current_node.value:
            return True
        elif key < current_node.value:
            return self._search(current_node.left, key)
        else:
            return self._search(current_node.right, key)



bst = BinarySearchTree()
bst.insert(51)
bst.insert(13)
bst.insert(20)
bst.insert(43)
bst.insert(70)
bst.insert(67)
bst.insert(80)

print(bst.search(13))
```

Output:

```
ces&Technol
True
```