GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

RECORD BOOK

GOVERNMENT OF KERALA

DEPARTMENT OF TECHNICAL EDUCATION

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(GOVT. ENGINEERING COLLEGE)

KOTTAYAM - 686501

20MCA241 DATA SCIENCE LAB

Name: PALLAVI K

Branch: Master of Computer Applications

Semester: 3

Roll No: 43

CERTIFIED BONAFIDE RECORD WORK DONE BY

Reg No. ....

STAFF IN CHARGE

INTERNAL EXAMINER **EXTERNAL EXAMINER**

Contents

Assignment 1
Review of python programming

Problem Statement

Write Python code to explore and practice with the basic data types, containers, functions, and classes of Python.

1. Start by creating variables of various numeric data types and assigning them values.
2. Print the data types and values of these variables.
3. Perform mathematical operations on these variables.
4. Update the values of these variables.
5. Create boolean variables with True or False values.
6. Print the data types of these boolean variables.
7. Perform Boolean operations on these boolean variables.
8. Create string variables with text values.
9. Print the contents and lengths of these string variables.
10. Concatenate strings.
11. Format strings with variables.
12. Use string methods to manipulate strings by capitalizing, converting to uppercase, justifying, centering, replacing substrings, and stripping whitespace.
13. Create and use Python lists. Perform tasks like appending elements, indexing, slicing, and iterating through the list.
14. Create and use Python tuples. Perform tasks like indexing, slicing, and concatenation.
15. Create and use Python sets. Perform tasks like accessing, adding, deleting set elements.
16. Create and use Python dictionaries. Perform tasks like adding, updating, and removing key-value pairs, and accessing values.
17. Define simple functions with parameters and return values.
18. Call functions with different arguments and use the returned results.
19. Write functions that accept other functions as arguments.

20. Define and use Python classes. Include tasks like creating a class, defining methods, and creating instances.
21. Implement class inheritance and method overriding.
22. Create a class with class variables and instance variables, and demonstrate their usage.

1.1 Basic data types

1.1.1 Numbers

```python
a=int(input("enter a intiger : "))
b=float(input("enter a floating number :"))
print(type(a))
print(type(b))
sum=a+b
sub=a-b
mul=a*b
div=a/b
mod=a%b
print("Sum=",sum)
print("Difference=",sub)
print("product=",mul)
print("division=",div)
print("mod=",mod)
a=5
b=3
print("a =",a)
print("b =",b)
```

```
enter a intiger : 2
enter a floating number : 2.3
<class 'int'>
<class 'float'>
Sum= 4.3
Difference= -0.2999999999999998
product= 4.6
division= 0.8695652173913044
mod= 2.0
a = 5
b = 3
```

1.1.2 Booleans

```python
c=True
d=False
print(type(c))
print(type(d))
print(c and d) # Logical AND;
print(c or d)# Logical OR;
print( not d)   # Logical NOT;
print(c!=d)  # Logical XOR;
```

```
<class 'bool'>
<class 'bool'>
False True True True
```

1.1.3 Strings

```python
    s="Hi my name is Pallavi"
    st="This is my first ds lab"
    print("Length of first string variable =",len(s))
    print(s+ "," + st)
    name="Pallavi"
    age=22
    formated_str="My name is {} and i'm {}".format(name,age)
    print(formated_str)
```

```
Hi my name is Pallavi
This is my first ds lab
Length of first string variable = 21
Hi my name is Pallavi,This is my first ds lab
My name is Pallavi and i'm 22
```

```python
s='hello'
print(s.capitalize())
print(s.upper())
print(s.lower())
print(s.rjust(7))
print(s.center(15))
print(s.replace('l','(ell)'))
print(' world'.strip())
```

```
Hello
HELLO
hello
  hello
     hello
he(ell)(ell)o
world
```

1.2 Containers

1.2.1 Lists

```
lis=[1,2,3,4]
print(lis)
lis.append(5)
print(lis)
lis[2] = 'hai'
print(lis)
print(lis[3])
r = lis.pop()
print(r,lis)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5]
[1, 2, 'hai', 4, 5]
4
5 [1, 2, 'hai', 4]
```

1.2.2 Slicing

```
n=list(range(5))
print(n)
print(n[1:3])
print(n[3:])
print(n[:3])
print(n[:-1])
n[2:4]=[8,9]
print(n)
```

```
[0, 1, 2, 3, 4]
[1, 2]
[3, 4]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 8, 9, 4]
```

1.2.3 Loops

```
li=[1,2,3,4]
for i in li:
    print(i)
```

```
1
2
3
4
```

1.2.4 List comprehensions

```python
num = [ 2, 3, 4, 6]
sq = []
for i in num:
    sq.append(i ** 2)
print(sq)
```

```
[4, 9, 16, 36]
```

1.2.5 Dictionaries

```python
v= {'car': 'Toyota', 'bike': 'splender'}
print(v)
print(v['car'])
print('car' in v)
v['jeep'] = 'mahindra'
print(v)
print(v['jeep'])
```

```
{'car': 'Toyota', 'bike': 'splender'}
Toyota
True
{'car': 'Toyota', 'bike': 'splender', 'jeep': 'mahindra'}
mahindra
```

1.2.6 Sets

```python
set1 = {'apple', 'banana', 'carrot'}
print(set1)
print('apple' in set1)
print('mango' in set1)
set1.add('mango')
print(len(set1))
set1.remove('banana')
print(set1)
print(len(set1))
```

```
{'carrot', 'banana', 'apple'}
True
False
4
{'mango', 'carrot', 'apple'}
3
```

1.2.7 Tuples

```python
t=(1,2,3,4)
t1=(6,7)
print(type(t))
print(t)
print(t[3])
print(t[1:3])
print(t+t1)
```

```
<class 'tuple'>
(1, 2, 3, 4)
4
(2, 3)
(1, 2, 3, 4, 6, 7)
```

1.3 Functions

```python
def sign(x):
    if x > 0:
        return 'positive'
    elif x < 0:
        return 'negative'
    else:
        return 'zero'
for x in [-1, 0, 1]:
    print(sign(x))
```

```
negative
zero
positive
```

```python
def hello(name, loud=False):
    if loud:
        print("HELLO, {}".format(name.upper()))
    else:
         print("Hello, {}!".format(name))
hello("Pallavi")
hello("Jack", loud=True)
```

```
Hello, Pallavi!
HELLO, Jack
```

```python
def apply_function(func, value):
    return func(value)
def square(x):
    return x * x
def cube(x):
    return x * x * x
print(apply_function(square, 3))
print(apply_function(cube, 3))
```

9

27

1.4 Classes

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display_details(self):
        print(f"Name: {self.name}, Age: {self.age}")
    def update_age(self, new_age):
        self.age = new_age
        print(f"{self.name}'s age has been updated to {self.age}")
person1 = Person("Pallavi", 22)
person2 = Person("Dayal", 21)
person1.display_details()
person2.display_details()
person1.update_age(23)
person1.display_details()
```

```
Name: Pallavi, Age: 22
Name: Dayal, Age: 21
Pallavi's age has been updated to 23
Name: Pallavi, Age: 23
```

1.4.1 inheritance and method overriding

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        print(f"{self.name} makes a sound.")
class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed
    def speak(self):
        print(f"{self.name} barks.")
generic_animal = Animal("Animal")
generic_animal.speak()
my_dog = Dog("Jacky", "Golden Retriever")
my_dog.speak()
```

```
Animal makes a sound.
Jacky barks.
```

1.4.2 class variables and instance variables

```
class MyClass:
    class_var = "I am a class variable"
    def _init_(self, instance_var):
        self.instance_var = instance_var
obj = MyClass("I am an instance variable")
print(MyClass.class_var)
print(obj.class_var)
print(obj.instance_var)
```

```
I am a class variable
I am a class variable
I am an instance variable
```

Assignment 2
Vectorized Computations using Numpy

Problem Statement

Implement the following computations using NumPy:

1. Create a matrix U of shape (m, n) with input values where m and n are input positive integers.
2. Compute X as the transpose of U.
3. Create a matrix Y of shape $(1, m)$ with random values $\in [0, 1]$.
4. Create a matrix $W1$ of shape (p, n) with random values $\in [0, 1]$ where p is an input positive integer.
5. Create a vector $B1$ of shape $(p, 1)$ with random values $\in [0, 1]$.
6. Create a vector $W2$ of shape $(1, p)$ with all zeros.
7. Create a scalar $B2$ with a random value $\in [0, 1]$.
8. Perform the following computations iteratively 15 times:
   (a) $Z1 = W1 \cdot X + B1$ (Matrix Multiplication)
   (b) $A1 = f(Z1)$ where f is a function that returns 0 for negative values and the input value itself otherwise.
   (c) $Z2 = W2 \cdot A1 + B2$
   (d) $A2 = g(Z2)$ where g is a function defined as $g(x) = \frac{1}{1+e^{-x}}$.
   (e) $L = \frac{1}{2}(A2 - Y)^2$
   (f) $dA2 = A2 - Y$
   (g) $dZ2 = dA2 \circ gprime(Z2)$ where $gprime(x)$ is a function that returns $g(x) \cdot (1 - g(x))$ and $\circ$ indicates element-wise multiplication.
   (h) $dA1 = W2^T \cdot dZ2$
   (i) $dZ1 = dA1 \circ fprime(Z1)$ where $fprime$ is a function that returns 1 for positive values and 0 otherwise and $\circ$ indicates element-wise multiplication.
   (j) $dW1 = \frac{1}{m} \cdot dZ1 \cdot X^T$
   (k) $dB1 = \frac{1}{m} \sum dZ1$ (sum along the columns)
   (l) $dW2 = \frac{1}{m} \cdot dZ2 \cdot A1^T$

(m) $dB2 = \frac{1}{m}\sum dZ2 \quad$ (sum along the columns)

(n) Update and print $W1$, $B1$, $W2$, and $B2$ for $\alpha = 0.01$:

i. $W1 = W1 - \alpha \cdot dW1$

ii. $B1 = B1 - \alpha \cdot dB1$

iii. $W2 = W2 - \alpha \cdot dW2$

iv. $B2 = B2 - \alpha \cdot dB2$

1.1 Matrix

1.1.1 Create Matrix

```
import numpy as np
m=int(input("Enter row size:"))
n=int(input("Enter column size:"))
print("Enter values in single line(space seperated formst):")
entries=list(map(int,input().split()))
U=np.array(entries).reshape(m,n)
print(U)
```

```
Enter row size:3
Enter column size:3
Enter values in single line(space seperated formst):
1 2 3 4 5 6 7 8 9
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

1.1.2 Matrix Transpose

```
X=U.T
print("Transpose of matrix U:")
print(X)
```

```
Transpose of matrix U:
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

1.1.3 Matrix with random values

```
m=5
Y=np.random.rand(1,m)
print(Y)

p=int(input("Enter  positive value,p :"))
n=3
W1=np.random.rand(p,n)
print("Matrix W1\n",W1)

p=int(input("Enter a value for p :"))
B1=np.random.rand(p,1)
print("Vector B1:")
print(B1)
```

```
[[0.77600842 0.7988252  0.56614616]]
Enter  positive value,p :3
Matrix W1
 [[0.41974795 0.92028813 0.2837073 ]
 [0.65957422 0.77089493 0.92210491]
 [0.38261143 0.09683981 0.5598406 ]]
Enter a value for p :3
Vector B1:
[[0.47381882]
 [0.03841201]
 [0.63450281]]
```

1.1.4 Matrix with Zeros

```
p=int(input("Enter value for p :"))
W2=np.zeros((1,p))
print(W2)
```

```
Enter value for p :3
[[0. 0. 0.]]
```

1.1.5 Scalar matrix

```
B2=np.random.rand()
print("Scalar B2:")
print(B2)
```

```
Scalar B2:
0.1903661347100598
```

1.1.6 Matrix Computations

```
import numpy as np

def f(Z):
    return np.maximum(0,Z)

def g(Z):
    return 1/(1+np.exp(-Z))

def gprime(Z):
    gz=g(Z)
    return gz*(1-gz)

def fprime(Z):
    return (Z > 0).astype(float)

alpha=0.01
for i in range(15):
    Z1=np.dot(W1,X) + B1

    A1=f(Z1)

    Z2=np.dot(W2,A1) + B2

    A2=g(Z2)

    L=0.5 * np.square(A2 - Y)

    dA2=A2 -Y

    dZ2=dA2 * gprime(Z2)

    dA1=np.dot(W2.T,dZ2)

    dZ1=dA1 * fprime(Z1)

    dW1=np.dot(dZ1, X.T)
```

```python
    dB1=np.sum(dZ1, axis=1,keepdims=True)/m

    dW2=np.dot(dZ2,A1.T)/m

    dB2=np.sum(dZ2)/m

    W1 -=alpha * dW1
    B1 -=alpha * dB1
    W2 -=alpha * dW2
    B2 -=alpha * dB2

print("\n Updated W1:\n",W1)
print("\n Updated B1:\n",B1)
print("\n Updated W2:\n",W2)
print("\n Updated B2:\n",B2)
print("\n Loss L:\n",L)
```

```
Updated W1:
 [[0.41980698 0.92043646 0.28394493]
 [0.65965781 0.77110449 0.92244042]
 [0.38265528 0.09695168 0.56002049]]

 Updated B1:
 [[0.47384859]
 [0.038454  ]
 [0.63452548]]

 Updated W2:
 [[0.01420082 0.01989279 0.01124518]]

 Updated B2:
 0.19398590364048363

 Loss L:
 [[0.01686055 0.0112311  0.00917792]]
```

Assignment 3
Vectorized Computations using TensorFlow

Problem Statement

Implement the following computations using TensorFlow::

1. Compute X as the transpose of U.
2. Create a matrix Y of shape $(1, m)$ with random values $\in [0, 1]$.
3. Create a matrix $W1$ of shape (p, n) with random values $\in [0, 1]$ where p is an input positive integer.
4. Create a vector $B1$ of shape $(p, 1)$ with random values $\in [0, 1]$.
5. Create a vector $W2$ of shape $(1, p)$ with all zeros.
6. Create a scalar $B2$ with a random value $\in [0, 1]$.
7. Perform the following computations iteratively 15 times:
   (a) $Z1 = W1 \cdot X + B1 \quad$ (Matrix Multiplication)
   (b) $A_1 = \text{ReLU}(Z_1)$
   where $\text{ReLU}(x)$ is a function that returns 0 for negative values and the input value itself otherwise.
   (c) $Z_2 = W_2 \cdot A_1 + B_2$
   (d) $A_2 = \text{softmax}(Z_2)$
   where $\text{softmax}(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
   (e) $dZ_2 = A_2 - \text{one hot } Y$
   where one hot Y is the one-hot encoded form of Y.
   (f) $dA_2 = W_2^T \cdot dZ_2$
   (g) $dW_2 = \frac{1}{m} \cdot dZ_2 \cdot A_1^T$
   (h) $dB_2 = \frac{1}{m} \cdot \sum \text{dZ}_2$
   (sum along the columns)
   (i) $dZ_1 = dA_2 \circ \text{ReLU deriv}(Z_1)$
   where $\text{ReLU deriv}(x)$ returns 1 for positive values and 0 otherwise, and $\circ$ indicates element-wise multiplication.
   (j) $dA_1 = W_1^T \cdot dZ_1$

(k) $dB_1 = \frac{1}{m} \cdot \sum \mathrm{dZ}_1$
(sum along the columns)

(l) $dW_1 = \frac{1}{m} \cdot dZ_1 \cdot X^T$

(m) Update and print W_1, B_1, W_2, and B_2 for $\alpha = 0.01$:

i. $W_1 = W_1 - \alpha \cdot dW_1$

ii. $B_1 = B_1 - \alpha \cdot dB_1$

iii. $W_2 = W_2 - \alpha \cdot dW_2$

iv. $B_2 = B_2 - \alpha \cdot dB_2$

1.1 Matrix

1.1.1 CREATE MATRIX

```
import numpy as np
import tensorflow as tf
def create_matrix(m,n):
u=tf.Variable(tf.random.normal(shape=(m,n)))
return u
m=int(input("enter the number of rows:"))
n=int(input("enter the number of columns:"))
matrix=create_matrix(m,n)
print(matrix.numpy())
```

```
enter the number of rows:3
enter the number of columns:4

[[ 0.23972483  0.50137824  0.7238233   2.3607743 ]
 [ 1.628231   -0.13696817 -0.9873514   0.6208397 ]
 [ 0.41659972  0.12066048  1.313003    1.486488  ]]
```

1.1.2 MATRIX TRANSPOSE

```
x=tf.transpose(matrix)
print(x.numpy())
```

```
[[ 0.23972483  1.628231    0.41659972]
 [ 0.50137824 -0.13696817  0.12066048]
 [ 0.7238233  -0.9873514   1.313003  ]
 [ 2.3607743   0.6208397   1.486488  ]]
```

1.1.3 MATRIX WITH RANDOM VALUES

```
y=tf.Variable(tf.random.uniform(shape=(1,m),minval=0,maxval=10,dtype=tf.int32)
    )
print(y.numpy())

p=int(input("enter the number of rows for w1:"))
w1=tf.Variable(tf.random.uniform(shape=(p,n),minval=0,maxval=1,dtype=tf.
    float32))
print(w1.numpy())

B1=tf.Variable(tf.random.uniform(shape=(p,1),minval=0,maxval=1,dtype=tf.
    float32))
print(B1.numpy())
```

```
[[9 7 6]]


enter the number of rows for w1:3
[[0.16307628 0.00463021 0.20726764 0.10553467]
 [0.38555396 0.51831377 0.78707254 0.07831943]
 [0.10433352 0.640242   0.33624303 0.23005724]]

[[0.66026664]
 [0.03466427]
 [0.28998554]]
```

1.1.4 MATRIX WITH ZEROS

```
w2=tf.Variable(tf.zeros(shape=(10,p)))
print(w2.numpy())
```

```
[[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]
[0. 0. 0.]]
```

1.1.5 SCALAR MATRIX

```
B2=tf.Variable(tf.random.uniform([],minval=-1,maxval=2,dtype=tf.float32))
print(B2.numpy())
```

0.79959846

1.1.6 MATRIX COMPUTATIONS

```
alpha=0.01
def relu(x):
    return tf.maximum(0,x)
def relu_deriv(x):
    return tf.where(x>0,tf.ones_like(x),tf.zeros_like(x))
def softmax(x):
    return tf.nn.softmax(x,axis=0)
def one_hot_encode(y,depth):
    return tf.one_hot(y,depth)
for _ in range(15):
    with tf.GradientTape() as tape:
        Z1 = tf.matmul(w1, x) + B1
        A1 = tf.nn.relu(Z1)
        Z2 = tf.matmul(w2, A1) + B2
        A2 = tf.nn.softmax(Z2)
        y_one_hot = one_hot_encode(y, 10)
        y_one_hot = tf.transpose(y_one_hot, perm=[2, 1, 0])
        y_one_hot = tf.reshape(y_one_hot, [10, 3])
        L = 0.5 * tf.reduce_sum(tf.square(A2 - tf.cast(y_one_hot, tf.float32)))
        dZ2 = A2 - tf.cast(y_one_hot, dtype=tf.float32)
        dA2 = tf.matmul(w2, dZ2, transpose_a=True)
        dW2 = (1/m) * tf.matmul(dZ2, A1, transpose_b=True)
        dB2 = (1/m) * tf.reduce_sum(dZ2, axis=1, keepdims=True)
        dZ1 = dA2 * tf.cast(Z1 > 0, dtype=tf.float32)
        dA1 = tf.matmul(w1, dZ1, transpose_a=True)
        dB1 = (1/m) * tf.reduce_sum(dZ1, axis=1, keepdims=True)
        dW1 = (1/m) * tf.matmul(dZ1, x, transpose_b=True)

    gradients = tape.gradient(L, [w1, B1, w2, B2])
    dW1, dB1, dW2, dB2 = gradients
    w1.assign_sub(alpha * dW1)
    B1.assign_sub(alpha * dB1)
    w2.assign_sub(alpha * dW2)
    B2.assign_sub(alpha * dB2)

    print(f"Updated W1:\n{w1.numpy()}")
    print(f"Updated B1:\n{B1.numpy()}")
    print(f"Updated W2:\n{w2.numpy()}")
    print(f"Updated B2:\n{B2.numpy()}")
```

```
Updated W1:
[[0.16274047 0.00473522 0.2078146  0.10584786]
 [0.38590193 0.51855755 0.7881588  0.08005956]
 [0.10318887 0.64067984 0.33789188 0.231299  ]]
Updated B1:
[[0.66026664]
 [0.03564009]
 [0.28998554]]
Updated W2:
[[ 4.5374473e-09  3.8035157e-09  4.1739439e-09]
 [ 4.5374473e-09  3.8035157e-09  4.1739439e-09]
 [ 4.5374473e-09  3.8035157e-09  4.1739439e-09]
 [ 4.5374473e-09  3.8035157e-09  4.1739439e-09]
 [ 4.5374473e-09  3.8035157e-09  4.1739439e-09]
 [ 4.5374473e-09  3.8035157e-09  4.1739439e-09]
 [ 7.0324750e-03  2.7476847e-02  1.2598041e-02]
 [-1.1448847e-02 -4.2491313e-02 -3.7542023e-02]
 [ 4.5374473e-09  3.8035157e-09  4.1739439e-09]
 [ 4.1745570e-03  1.4119232e-02  2.4135472e-02]]
Updated B2:
0.7995984554290771
```