```
a='123'
print(type(a))
b=float(a)
print(type(b))
#%%
s1="HELLO"
s2="hello"

output:-

<class 'str'>
<class 'float'>

[Program finished]

# -*- coding: utf-8 -*-
"""

chapter :- 9


def my_bin_2_dec(b):
    d=0
    for digit in b:
        d=d*2 +int(digit)
        return d;
#%%
def my_dec_2_bin(d):
    b=[]
    if d==0:
        b.append(0)
    while d>= 1:
            b.append(d%2)
            d= d//2
    b.reverse()
    return b;
#%%
def my_dec_2_bin(d):
    b=[]
    if d==0:
        b.append(0)
        while d>= 1:
            b.append(d%2)
            d= d//2
            b.reverse()
            return b;

def my_bin_2_dec(b):
    d=0
    for digit in b:
```

```
        d=d*2 +int(digit)
        return d;
#%%
def my_bin_adder (b1, b2):
    max_len=max(len(b1), len(b2))
    b1= [0]*(max_len- len(b1)) +b1
    b2= [0]*(max_len- len(b2)) +b2
    b=[]
    carry=0
    for i in range(max_len-1, -1, -1):
        sum= carry +b1[i] +b2[i]
        res=1 if sum%2 ==1 else 0
        b.append(res)
        carry= 0 if sum <2 else 1

    if carry != 0:
        b.append(1)

    b.reverse()
    return b
```

chapter:- 11,12,13,15

```
@author: 91703_0zbjuu
"""
import numpy as np
from numpy.linalg import qr
a = np.array([[0, 2],
              [2, 3]])
q, r = qr(a)
print('Q:', q)
print('R:', r)
b = np.dot(q, r)
print('QR:', b)
a = np.array([[0, 2],
              [2, 3]])
p = [1, 5, 10, 20]
for i in range(20):
    q, r = qr(a)
    a = np.dot(r, q)
    if i+1 in p:
        print(f'Iteration {i+1}:')
        print(a)
#%%
import numpy as np
def normalize(x):
    fac = abs(x).max()
    x_n = x / x.max()
```

```
    return fac, x_n
x = np.array([1, 1, 1])
a=[[2,1,2],[1,3,2],[2,4,1]]
for i in range(8):
    x = np.dot(a, x)
    lambda_1, x = normalize(x)
    print("Eigenvalue:", lambda_1)
    print("Eigenvector:", x)
#%%
```

chapter :- 16

```
"""
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
A = np.vstack([x, np.ones(len(x))]).T
y = y[:, np.newaxis]
alpha = np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
print(alpha)
plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.')
plt.plot(x, alpha[0]*x + alpha[1], 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
pinv = np.linalg.pinv(A)
alpha = pinv.dot(y)
print(alpha)
alpha = np.linalg.lstsq(A, y, rcond=None)[0]
print(alpha)
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
def func(x, a, b):
    y = a*x + b
    return y
alpha = optimize.curve_fit(func, xdata = x, ydata = y)[0]
print(alpha)
#%%
import numpy as np
import matplotlib.pyplot as plt
x=np.array([0,1,2,3,4,5,6,7,8,9])
y=np.array([0,0.8,0.9,0.1,-0.6,-0.8,-1,-0.9,-0.4,2])
A=np.vstack([x,np.ones(len(x))]).T
plt.figure(figsize=(24,10))
```

```
i=2
plt.subplot(2,3,i)
plt.plot(x,y,"o")
plt.plot(x,y_est[0]*x**2+y_est[1]*x+0,"r",label="multivariant regression")
plt.title(f"Polynomial of order{i}Least square regression formula")
plt.legend()
plt.tight_layout()
plt.show()"""
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
A = np.vstack([x, np.ones(len(x))]).T
y = y[:, np.newaxis]
alpha = np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
print(alpha)
plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.')
plt.plot(x, alpha[0]*x + alpha[1], 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
pinv = np.linalg.pinv(A)
alpha = pinv.dot(y)
print(alpha)
alpha = np.linalg.lstsq(A, y, rcond=None)[0]
print(alpha)
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
def func(x, a, b):
    y = a*x + b
    return y
alpha = optimize.curve_fit(func, xdata = x, ydata = y)[0]
print(alpha)
#%%
import numpy as np
import matplotlib.pyplot as plt
x=np.array([0,1,2,3,4,5,6,7,8,9])
y=np.array([0,0.8,0.9,0.1,-0.6,-0.8,-1,-0.9,-0.4,2])
A=np.vstack([x,np.ones(len(x))]).T
plt.figure(figsize=(24,10))
i=2
plt.subplot(2,3,i)
plt.plot(x,y,"o")
plt.plot(x,y_est[0]*x**2+y_est[1]*x+0,"r",label="multivariant regression")
plt.title(f"Polynomial of order{i}Least square regression formula")
plt.legend()
```

```
plt.tight_layout()
plt.show()


import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
A = np.vstack([x, np.ones(len(x))]).T
y = y[:, np.newaxis]
alpha = np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
print(alpha)
plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.')
plt.plot(x, alpha[0]*x + alpha[1], 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
pinv = np.linalg.pinv(A)
alpha = pinv.dot(y)
print(alpha)
alpha = np.linalg.lstsq(A, y, rcond=None)[0]
print(alpha)
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
def func(x, a, b):
    y = a*x + b
    return y
alpha = optimize.curve_fit(func, xdata = x, ydata = y)[0]
print(alpha)
#%%
import numpy as np
import matplotlib.pyplot as plt
x=np.array([0,1,2,3,4,5,6,7,8,9])
y=np.array([0,0.8,0.9,0.1,-0.6,-0.8,-1,-0.9,-0.4,2])
A=np.vstack([x,np.ones(len(x))]).T
plt.figure(figsize=(24,10))
i=2
plt.subplot(2,3,i)
plt.plot(x,y,"o")
plt.plot(x,y_est[0]*x**2+y_est[1]*x+0,"r",label="multivariant regression")
plt.title(f"Polynomial of order{i}Least square regression formula")
plt.legend()
plt.tight_layout()
plt.show()import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
```

```
plt.style.use('seaborn-poster')
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
A = np.vstack([x, np.ones(len(x))]).T
y = y[:, np.newaxis]
alpha = np.dot((np.dot(np.linalg.inv(np.dot(A.T,A)),A.T)),y)
print(alpha)
plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.')
plt.plot(x, alpha[0]*x + alpha[1], 'r')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
pinv = np.linalg.pinv(A)
alpha = pinv.dot(y)
print(alpha)
alpha = np.linalg.lstsq(A, y, rcond=None)[0]
print(alpha)
x = np.linspace(0, 1, 101)
y = 1 + x + x * np.random.random(len(x))
def func(x, a, b):
    y = a*x + b
    return y
alpha = optimize.curve_fit(func, xdata = x, ydata = y)[0]
print(alpha)
#%%
import numpy as np
import matplotlib.pyplot as plt
x=np.array([0,1,2,3,4,5,6,7,8,9])
y=np.array([0,0.8,0.9,0.1,-0.6,-0.8,-1,-0.9,-0.4,2])
A=np.vstack([x,np.ones(len(x))]).T
plt.figure(figsize=(24,10))
i=2
plt.subplot(2,3,i)
plt.plot(x,y,"o")
plt.plot(x,y_est[0]*x**2+y_est[1]*x+0,"r",label="multivariant regression")
plt.title(f"Polynomial of order{i}Least square regression formula")
plt.legend()
plt.tight_layout()
plt.show()
```

chapter :- 18

```
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')
x = [0, 1, 2]
y = [1, 3, 2]
```

```
f = interp1d(x, y)
y_hat = f(1.5)
print(y_hat)
plt.figure(figsize = (10,8))
plt.plot(x, y, '-ob')
plt.plot(1.5, y_hat, 'ro')
plt.title('Linear Interpolation at x = 1.5')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
#%%
from scipy.interpolate import CubicSpline
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')
x = [0, 1, 2]
y = [1, 3, 2]

# use bc_type = 'natural' adds the constraints as we described above
f = CubicSpline(x, y, bc_type='natural')
x_new = np.linspace(0, 2, 100)
y_new = f(x_new)
plt.figure(figsize = (10,8))
plt.plot(x_new, y_new, 'b')
plt.plot(x, y, 'ro')
plt.title('Cubic Spline Interpolation')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
b = np.array([1, 3, 3, 2, 0, 0, 0, 0])
b = b[:, np.newaxis]
A = np.array([[0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 1, 1, 1], [1, 1, 1, 1, 0, 0, 0, 0], \
        [0, 0, 0, 0, 8, 4, 2, 1], [3, 2, 1, 0, -3, -2, -1, 0], [6, 2, 0, 0, -6, -2, 0, 0],\
        [0, 2, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 12, 2, 0, 0]])
np.dot(np.linalg.inv(A), b)
#%%
import numpy as np
import numpy.polynomial.polynomial as poly
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')
x = [0, 1, 2]
y = [1, 3, 2]
P1_coeff = [1,-1.5,.5]
P2_coeff = [0, 2,-1]
P3_coeff = [0,-.5,.5]

# get the polynomial function
```

```
P1 = poly.Polynomial(P1_coeff)
P2 = poly.Polynomial(P2_coeff)
P3 = poly.Polynomial(P3_coeff)

x_new = np.arange(-1.0, 3.1, 0.1)

fig = plt.figure(figsize = (10,8))
plt.plot(x_new, P1(x_new), 'b', label = 'P1')
plt.plot(x_new, P2(x_new), 'r', label = 'P2')
plt.plot(x_new, P3(x_new), 'g', label = 'P3')

plt.plot(x, np.ones(len(x)), 'ko', x, np.zeros(len(x)), 'ko')
plt.title('Lagrange Basis Polynomials')
plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.legend()
plt.show()
L = P1 + 3*P2 + 2*P3

fig = plt.figure(figsize = (10,8))
plt.plot(x_new, L(x_new), 'b', x, y, 'ro')
plt.title('Lagrange Polynomial')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
#%%
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')

%matplotlib inline
def divided_diff(x, y):
    '''
    function to calculate the divided
    differences table
    '''
    n = len(y)
    coef = np.zeros([n, n])
    # the first column is y
    coef[:,0] = y

    for j in range(1,n):
        for i in range(n-j):
            coef[i][j] = \
           (coef[i+1][j-1] - coef[i][j-1]) / (x[i+j]-x[i])

    return coef
```

```
def newton_poly(coef, x_data, x):
    '''
    evaluate the newton polynomial
    at x
    '''
    n = len(x_data) - 1
    p = coef[n]
    for k in range(1,n+1):
        p = coef[n-k] + (x -x_data[n-k])*p
    return p
x = np.array([-5, -1, 0, 2])
y = np.array([-2, 6, 1, 3])
# get the divided difference coef
a_s = divided_diff(x, y)[0, :]

# evaluate on new data points
x_new = np.arange(-5, 2.1, .1)
y_new = newton_poly(a_s, x, x_new)

plt.figure(figsize = (12, 8))
plt.plot(x, y, 'bo')
plt.plot(x_new, y_new)
#%%
import numpy as np
import matplotlib.pyplot as plt
def my_nearest_neighbour(x0,y0,x):
    xi=np.abs(x_list-x0).argmin()
    print("xlist-x0=",(x_list-x0))
    print("x_list:",x_list,"\nxi:",xi)
    yi=np.abs(y_list-y0).argmin()
    print("ylist-y0=",(y_list-y0))
    print("y_list:",y_list,"\nyi:",yi)
    return data[xi,yi]
x_list=np.array([2.14,3.25,4.36,5.47,6.58])
y_list=np.array([3.65,5.86,7.47,5.99,6.8])
data=np.array([[1,0,1,0,1],[0,1,1,1,0],[1,1,0,0,0],[0,1,1,1,0],[1,1,1,0,0]])
print (data)
dat1=my_nearest_neighbour(4.1,5.9,x_list)
print ("data at (4.1,5.9)=:",dat1)
dat2=my_nearest_neighbour(6.7,4.1,x_list)
print ("data2 at(2.76,7.1)=:",dat2)
plt.plot(x_list,y_list,"ro")
plt.plot(x_list,y_list,"b")
plt.annotate("Point2",(4.1,5.9),size=20)
plt.plot(4.1,5.9,'ro',ms=15)
plt.annotate("Point2",(6.7,4.1),size=20)
plt.plot(6.7,4.1,'ro',ms=15)
plt.xlabel("x",size=20)
plt.ylabel("y",size=20)
```

```
plt.title("Nearest Neighbor Interpolitan",size=20)
plt.show()
#%%
import numpy as np
def my_double_exp(x, n):
    for i, j in zip(x, n):
        exp = 0
        var = i
        for order in range(j):
            exp = exp + (var)**(2*order)/np.math.factorial(order)
    print(f"Using first {j} terms for x = {i}, the approximation is {exp}")
    print(f"True value of e^2^2 is: {np.exp(2**2)}")
#%%
import math
import numpy as np
x=2
e_to_taylor=0
for i in range(7):
    e_to_taylor += x**i/math.factorial(i)
    print(f"Using {i}- term = {e_to_taylor}")
    print ("Actual value using Taylor Series= ",e_to_taylor)
    print ("\n")
    exp_py=math.exp(x)
    print ("Actual value Using math=",exp_py)
    print ("\n")
    exp_np=np.exp(2)
    for i in range(7):
        exp_np=np.exp(2)
        print(f"Using {i}- term = {exp_np}")
        print ("Actual Value Using numpy=",exp_np)
        print ("Truncation error is = ",abs(e_to_taylor-np.exp(2)))aa
```

lab chp:11

```
"""
num=int(input('enter your number: '))
for i in range(1,11):
    multiplied_no=num*i
    print(num,' X ',i , '= ', multiplied_n )
```

output :-

enter a number :5

5×1 = 5

5×2 = 10

5×3 =15

5×4 = 20