```
import numpy as np
```

Task1

a)Use above two procedures to find the second derivative of f(x)=x**2 exp(-x).

```
from sympy import *
x = symbols('x')
f = (x**2) * exp(-x)


df1 = diff(f, x, 2)
print(df1)
```

```
(x**2 - 4*x + 2)*exp(-x)
```

```
df2 = f.diff(x, 2)
print(df2)
```

```
(x**2 - 4*x + 2)*exp(-x)
```

b)Convert symbolic expression in part (a) into numpy function.

```
f_np = lambdify(x, f)
print(f_np)
```

```
<function _lambdifygenerated at 0x7ff251daf160>
```

c)Evaluate the numpy function (obtained in part b)at a single value and at an array.

```
#single value
my_choice = 2
print(f_np(my_choice))
```

```
0.5413411329464508
```

```
#using array
my_choice2 = np.array([1, 2, 3])
print(f_np(my_choice2))
```

```
[0.36787944 0.54134113 0.44808362]
```

Task 2: Write a code for Backward difference approximation (apply forward difference approximation on first point)

```
# code of backward difference formula.

import numpy as np
from tabulate import tabulate

def back_diff(x, y):

    # Compute the step size h
    h = x[1] - x[0]
    data=[]

    # Compute the backward difference approximation
    fdf = np.zeros_like(y)
    fdf[0] = (y[1] - y[0]) / h  # use forward difference on the first point
    data.append([x[0],y[0],fdf[0]])
    for i in range(len(y)-1):
        fdf[i+1] = (y[i+1] - y[i]) / h
        data.append([x[i+1],y[i+1],fdf[i+1]])

    print(tabulate(data,headers=['x','f(x)','df(x)/dx'],tablefmt="github"))

    return
```

```
# example to run above code
x=[0.2,0.4,0.6,0.8]
y=[3,3.9,3.98,4.2]
back_diff(x, y)
```

```
|   x |   f(x) |   df(x)/dx |
|-----|--------|------------|
| 0.2 |   3    |        4.5 |
| 0.4 |   3.9  |        4.5 |
| 0.6 |   3.98 |        0.4 |
| 0.8 |   4.2  |        1.1 |
```

Task 3: Make a code for five point endpoint and midpoint formulae where possible in given table.:

```
def five_pt(x, y):

    # Compute the step size h
    data=[]
    h = x[1] - x[0]

    # Compute the forward difference approximation
    tp = np.zeros_like(y)
    tp[0]=(-25*y[0]+48*y[1]-36*y[2]+16*y[3]-3*y[4])/(12*h) #five point endpoint (left end) formula
    tp[1]=(-25*y[1]+48*y[2]-36*y[3]+16*y[4]-3*y[5])/(12*h)
    tp[-1]=(25*y[-1]-48*y[-2]+36*y[-3]-16*y[-4]+3*y[-5])/(12*h) #five point endpoint (right end) formula
    tp[-2]=(25*y[-2]-48*y[-3]+36*y[-4]-16*y[-5]+3*y[-6])/(12*h)

    data.append([x[0],y[0],tp[0]])
    data.append([x[1],y[1],tp[1]])
    for i in range(2,len(y)-2):
        tp[i] = (y[i-2] -8*y[i-1]+8*y[i+1]-y[i+2]) / (12*h)
        data.append([x[i],y[i],tp[i]])
    data.append([x[-2],y[-2],tp[-2]])
    data.append([x[-1],y[-1],tp[-1]])

    print(tabulate(data,headers=['x','f(x)','df(x)/dx'],tablefmt="github"))


    return


x=[2.1,2.2,2.3,2.4,2.5,2.6]
y=[1.709847, 1.373823, 1.119214, 0.9160143, 0.7470223, 0.6015966]
five_pt(x, y)
```

```
|   x |     f(x) |   df(x)/dx |
|-----|----------|------------|
| 2.1 | 1.70985  |   -3.89934 |
| 2.2 | 1.37382  |   -2.87688 |
| 2.3 | 1.11921  |   -2.2497  |
| 2.4 | 0.916014 |   -1.83776 |
| 2.5 | 0.747022 |   -1.54421 |
| 2.6 | 0.601597 |   -1.3555  |
```

Task 4: Make a code of composite simpson's 1/3rd rule (set n=2 for simple simpson and raise exception when user enters n=odd value) and run on f(x) mentioned in exercise # 4.2, Question #5c and excercise # 4.3, Question 3e

```
def comp_simpson1_3rd_rule(f, a, b, n=1):  #n=1 indicates simple trpezoidal rule
     h = (b - a) / n
     x = [a + i*h for i in range(n+1)]
     y = [f(xi) for xi in x]
     s = sum(y[1:-1])
     ans=h/2 * (y[0] + 2*s + y[-1])
     return ans


def simpson_one_third_rule(f, a, b, n=2):  # n=2 indicates Simpson's 1/3 rule
    if n % 2 != 0:
        raise ValueError("n must be an even integer.")
    h = (b - a) / n
    x = [a + i*h for i in range(n+1)]
    y = [f(xi) for xi in x]
    s1 = sum(y[1:-1:2])
    s2 = sum(y[2:-2:2])
    ans = h/3 * (y[0] + 4*s1 + 2*s2 + y[-1])
```

```
    return ans


def f(x):
  return(2*x/(x**2-4))
simp = simpson_one_third_rule(f,1,1.6)
print(simp)
```

```
-0.7391053391053395
```

```
def f(x):
  return(x**2 * (ln(x)))

simp1 = simpson_one_third_rule(f,1,1.5)
print(simp1)
```

```
0.192245307413098
```

Task 5: Find Error bound for Exercise 4.3 Qno 7 part(a) and (b)

```
def Error_bound_simp(f,l,u):
   d4f = diff(f, x,4)
   abs_max_ddf=max(abs(d4f.subs(x,l)),abs(d4f.subs(x,u)))
   h=(u-l)/2
   Error_bound=h**5*abs_max_ddf/90
   return(Error_bound,abs_max_ddf)


x = symbols('x')
f = x**4
eb, _ = Error_bound_simp(f, 0.5, 1)
print(eb)
```

```
0.000260416666666667
```

```
f = 2/(x-4)
eb, _ = Error_bound_simp(f, 0, 0.5)
print(eb)
```

```
9.91650304436643e-7
```

✓ 0s completed at 12:24 AM