Q1

The code defines a function fun(x) that returns the sine of x^2 and another function derivative(x) that returns the derivative of fun(x) using the analytical formula. It then defines a function ForwardFinitDifferenceApp(x , h = 0.01) that calculates the forward finite difference approximation of the derivative of fun(x) using a small step size h.

The code also defines a function visualise() that generates a plot to compare the actual derivative and the forward finite difference approximation of the derivative for values of x ranging from 0 to 1. The plot is generated using the Matplotlib library and displays two lines, one representing the actual derivative and the other representing the approximation using the forward finite difference method.

Finally, the code checks if it is being executed as the main program and calls the visualise() function.

Q2

The code defines a function fun(x) that returns the sine of x^2 and another function derivative(x) that returns the derivative of fun(x) using the analytical formula. It then defines a function FinitDifferenceApp(x, h=0.01, type='f') that calculates the finite difference approximation of the derivative of fun(x) using a small step size h and the specified type of finite difference scheme, which can be 'f' for forward difference, 'b' for backward difference, or 'c' for central difference.

The code also defines a function visualise() that generates a plot to compare the absolute error of the finite difference approximation and the actual derivative for values of x ranging from 0 to 1 using the different types of finite difference schemes. The plot is generated using the Matplotlib library and displays three lines, one representing the absolute error of the forward difference, another representing the absolute error of the backward difference, and the third representing the absolute error of the central difference scheme.

Finally, the code checks if it is being executed as the main program and calls the visualise() function

Q4

The main function of the code is visualize, which takes two arguments a and b - the limits of integration, and performs the following steps:

Define a list x_axis to store the number of intervals used for approximation.

Define a list y_axis to store the computed area for each value of M (the number of intervals).

Compute the actual area under the curve using the function fun_integral.

For each value of M from 1 to 100, compute the area under the curve using the trapezoidal rule with M intervals.

Append the value of M to x_axis, and the computed area to y_axis.

Plot the values in x_axis and y_axis on a graph, with M on the x-axis and the computed area on the y-axis.

Also plot a horizontal line for the actual area, to compare with the computed values.

The function fun computes the function 2*x*exp(x*x), which is the function whose integral is approximated using the trapezoidal rule. The function fun_integral computes the integral of fun using the exact formula exp(x * x).

The graph generated by the code shows that as the number of intervals used for approximation is increased, the computed area approaches the actual area under the curve. This is expected, as the trapezoidal rule becomes more accurate with more intervals.