Q8:

```
import numpy as np

def Lambert_W(x, branch=0, tol=1e-8, init=1):
    # Check if the input arguments are consistent
    if x < 0:
        if branch == 0:
            print("Error: x < 0 and branch = 0. Choose branch = -1 instead.")
            return None, None
    if x >= np.exp(-1):
        if branch == -1:
            print("Error: x >= e^(-1) and branch = -1. Choose branch = 0 instead.")
            return None, None
    if tol <= 0 or tol > 1e-1:
        print("Error: Tolerance must be 0 < tol < 1e-1.")
        return None, None
    if init != 0 and init != 1:
        print("Error: init must be either 0 or 1.")
        return None, None

    # Define the maximum number of iterations
    itMax = 30


    # Initial guess for y_0
    if branch == 0 and init == 0:
        if x == -1 / np.exp(1):
            y = -1

        elif -1 / np.exp(1) < x < 0:
            y= (np.exp(1) * x / (1 + np.exp(1) * x + np.sqrt(1 + np.exp(1) * x))) * np.log(1 +
np.sqrt(1 + np.exp(1) * x))

        elif 0 < x < np.exp(1):
            y = x / np.exp(1)

        else:
            y = np.log(x) - np.log(np.log(x))

    if branch == -1 and init == 1:
        if x == -1 / np.exp(1):
            y = -1

        elif -1 / np.exp(1) < x < -1 / 4:
            y = -1 - np.sqrt(2 * (1 + np.exp(1) * x))

        elif -1 / 4 < x < 0:
            y = np.log(-x) - np.log(-np.log(-x))


    # Implement Halley's method
    it = 0
    while it < itMax:
        f = y * np.exp(y) - x
        f_prime = np.exp(y) * (y + 1)
        f_double_prime = np.exp(y) * (2 * y + 2)
        y_new = y - 2 * f * f_prime / (2 * f_prime**2 - f * f_double_prime)

        # Check if the tolerance is reached
        if abs(y_new - y) <= tol:
            break
```

```
        y = y_new
        it += 1

    if it == itMax:
        print("Warning: Maximum number of iterations reached. Tolerance may not be achieved.")

    return y, it

# Evaluate the omega constant
omega, it = Lambert_W(1, branch=0, tol=1e-8, init=0)
print(f"Omega obtained via calculation:{omega}\nDiffrence between the omega obtained in this calculation and Wikipedia value:{abs(omega - 0.56714329040978387299968662210)}")
print("Number of iterations:", it)
```

Answer:

```
Omega obtained via calculation:0.567143290389849
Diffrence between the omega obtained in this calculation and Wikipedia value:1.9934831563261923e-11
Number of iterations: 3
```