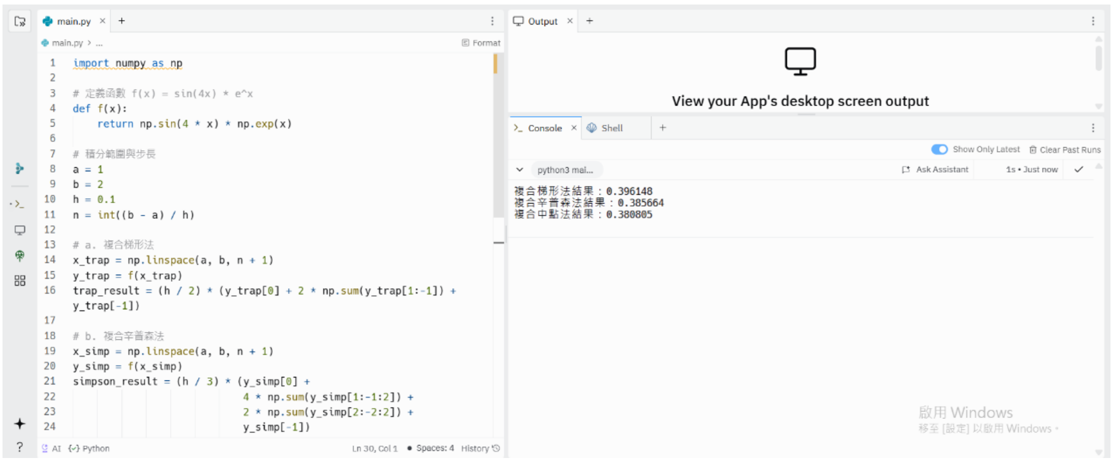
Q1

a. the composite trapezoidal rule ：0.396148

b. the composite Simpsons’ method ：0.385664

c the composite midpoint rule：0.380805



```python
import numpy as np

# 定義函數 f(x) = sin(4x) * e^x
def f(x):
    return np.sin(4 * x) * np.exp(x)

# 積分範圍與步長
a = 1
b = 2
h = 0.1
n = int((b - a) / h)

# a. 複合梯形法
x_trap = np.linspace(a, b, n + 1)
y_trap = f(x_trap)
trap_result = (h / 2) * (y_trap[0] + 2 * np.sum(y_trap[1:-1]) +
y_trap[-1])

# b. 複合辛普森法
x_simp = np.linspace(a, b, n + 1)
y_simp = f(x_simp)
simpson_result = (h / 3) * (y_simp[0] +
                            4 * np.sum(y_simp[1:-1:2]) +
                            2 * np.sum(y_simp[2:-2:2]) +
                            y_simp[-1])
```

```
複合梯形法結果：0.396148
複合辛普森法結果：0.385664
複合中點法結果：0.380805
```

Q2

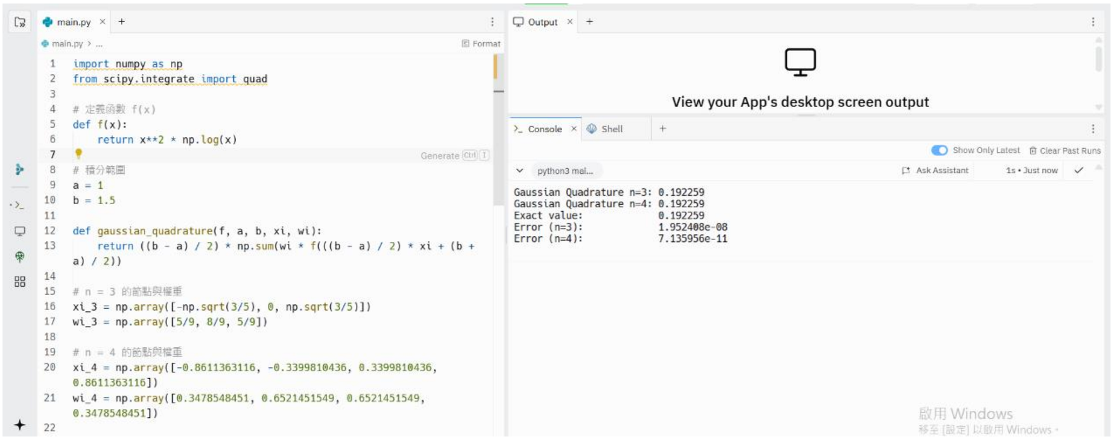
Gaussian Quadrature n=3: 0.192259

Gaussian Quadrature n=4: 0.192259

Exact value: 0.192259

Error (n=3): 1.952408e-08

Error (n=4): 7.135956e-11



```python
import numpy as np
from scipy.integrate import quad

# 定義函數 f(x)
def f(x):
    return x**2 * np.log(x)

# 積分範圍
a = 1
b = 1.5

def gaussian_quadrature(f, a, b, xi, wi):
    return ((b - a) / 2) * np.sum(wi * f(((b - a) / 2) * xi + (b +
a) / 2))

# n = 3 的節點與權重
xi_3 = np.array([-np.sqrt(3/5), 0, np.sqrt(3/5)])
wi_3 = np.array([5/9, 8/9, 5/9])

# n = 4 的節點與權重
xi_4 = np.array([-0.8611363116, -0.3399810436, 0.3399810436,
0.8611363116])
wi_4 = np.array([0.3478548451, 0.6521451549, 0.6521451549,
0.3478548451])
```

```
Gaussian Quadrature n=3: 0.192259
Gaussian Quadrature n=4: 0.192259
Exact value:             0.192259
Error (n=3):             1.952408e-08
Error (n=4):             7.135956e-11
```

Q3

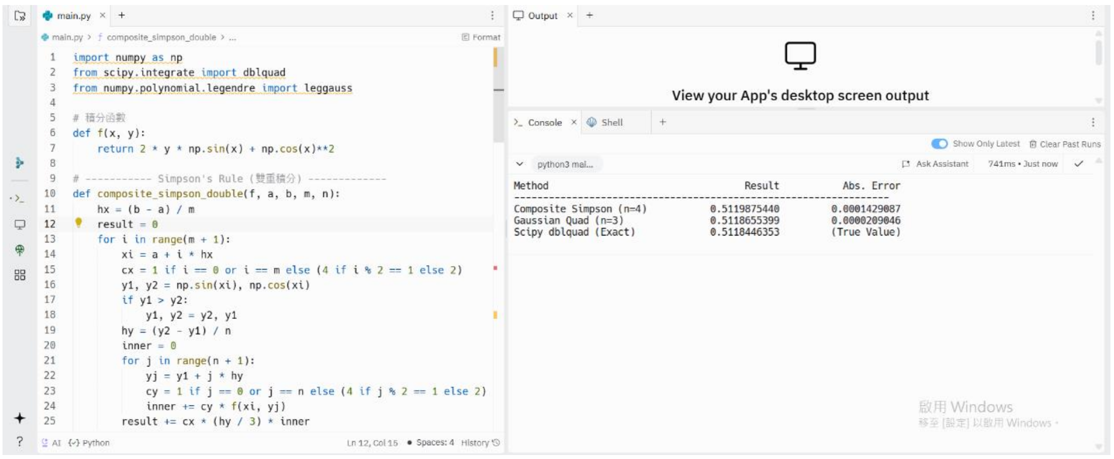
Composite Simpson (n=4): 0.5119875440

Gaussian Quad (n=3) : 0.5118655399

Exact value: 0.5118446353

Error (Composite Simpson): 0.0001429087

Error(Gaussian Quad): 0.0000209046



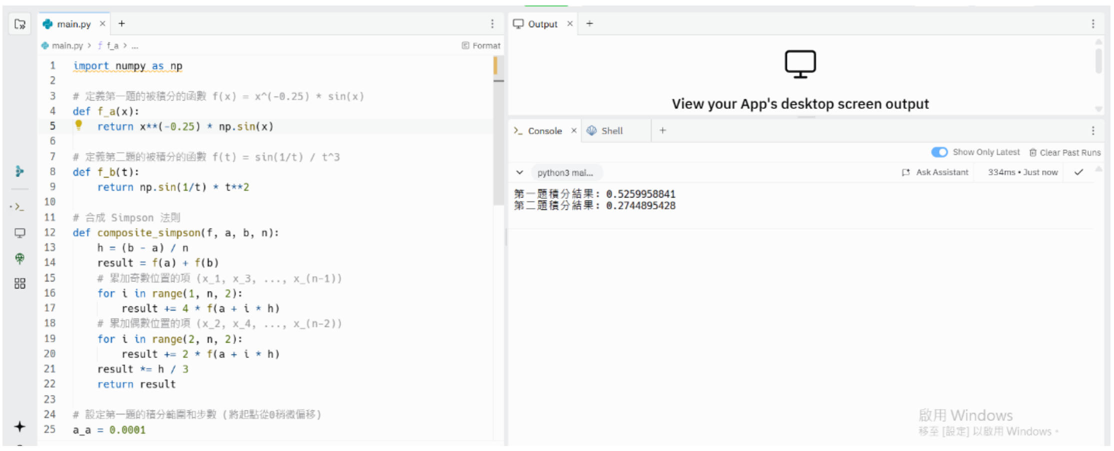
```python
import numpy as np
from scipy.integrate import dblquad
from numpy.polynomial.legendre import leggauss

# 積分函數
def f(x, y):
    return 2 * y * np.sin(x) + np.cos(x)**2

# ---------- Simpson's Rule (雙重積分) ------------
def composite_simpson_double(f, a, b, m, n):
    hx = (b - a) / m
    result = 0
    for i in range(m + 1):
        xi = a + i * hx
        cx = 1 if i == 0 or i == m else (4 if i % 2 == 1 else 2)
        y1, y2 = np.sin(xi), np.cos(xi)
        if y1 > y2:
            y1, y2 = y2, y1
        hy = (y2 - y1) / n
        inner = 0
        for j in range(n + 1):
            yj = y1 + j * hy
            cy = 1 if j == 0 or j == n else (4 if j % 2 == 1 else 2)
            inner += cy * f(xi, yj)
        result += cx * (hy / 3) * inner
```

```
Method                          Result          Abs. Error
--------------------------------------------------------------
Composite Simpson (n=4)         0.5119875440    0.0001429087
Gaussian Quad (n=3)             0.5118655399    0.0000209046
Scipy dblquad (Exact)           0.5118446353    (True Value)
```

Q4

第一題: 0.5259958841

第二題: 0.2744895428



```python
import numpy as np

# 定義第一題的被積分的函數 f(x) = x^(-0.25) * sin(x)
def f_a(x):
    return x**(-0.25) * np.sin(x)

# 定義第二題的被積分的函數 f(t) = sin(1/t) / t^3
def f_b(t):
    return np.sin(1/t) * t**2

# 合成 Simpson 法則
def composite_simpson(f, a, b, n):
    h = (b - a) / n
    result = f(a) + f(b)
    # 累加奇數位置的項 (x_1, x_3, ..., x_(n-1))
    for i in range(1, n, 2):
        result += 4 * f(a + i * h)
    # 累加偶數位置的項 (x_2, x_4, ..., x_(n-2))
    for i in range(2, n, 2):
        result += 2 * f(a + i * h)
    result *= h / 3
    return result

# 設定第一題的積分範圍和步數 (將起點從0稍微偏移)
a_a = 0.0001
```

```
第一題積分結果: 0.5259958841
第二題積分結果: 0.2744895428
```