數值 hw4

1.

```
import numpy as np

def f(x):
    return np.exp(x) * np.sin(4*x)

def composite_trapezoidal(a, b, h):
    n = int((b - a) / h)
    x = np.linspace(a, b, n + 1)
    y = f(x)
    return (h / 2) * (y[0] + 2 * np.sum(y[1:n]) + y[n])

def composite_simpson(a, b, h):
    n = int((b - a) / h)
    if n % 2 == 1:
        n += 1  # Simpson's rule requires an even number of intervals
    x = np.linspace(a, b, n + 1)
    y = f(x)
    return (h / 3) * (y[0] + 4 * np.sum(y[1:n:2]) + 2 * np.sum(y[2:n-1:2]) + y[n])

def composite_midpoint(a, b, h):
```

input

```
Composite Trapezoidal Rule: 0.396148
Composite Simpson's Rule: 0.385664
Composite Midpoint Rule: 0.380805

...Program finished with exit code 0
```

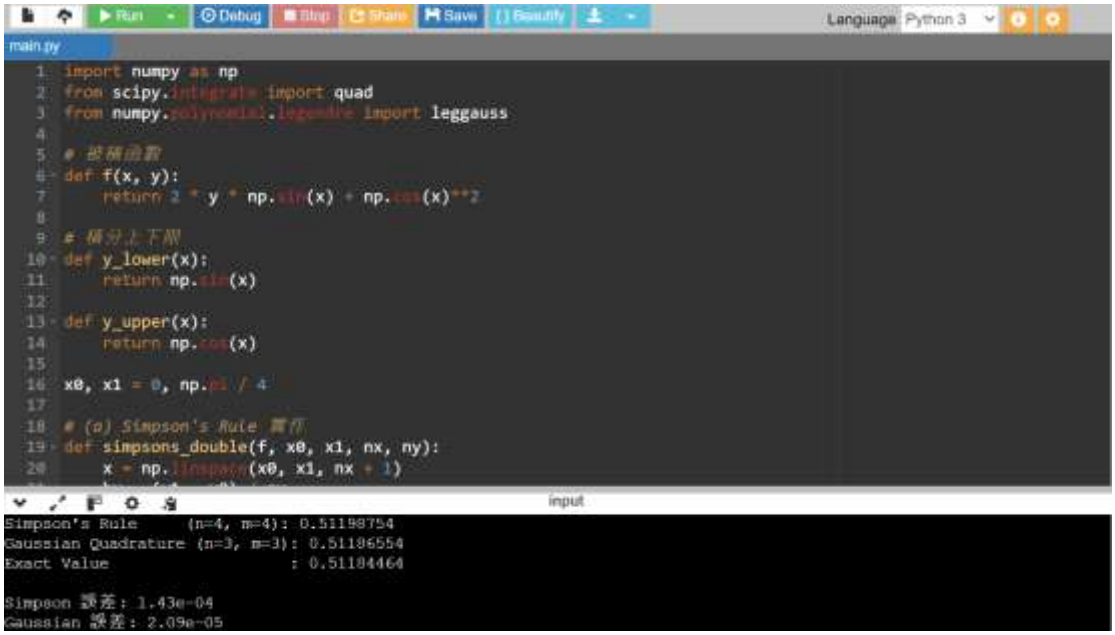
2.

```
import numpy as np
import scipy.integrate as spi

def f(x):
    return x**2 * np.log(x)

def gauss_quadrature(a, b, n):
    # 取得勒讓德多項式的根與權重
    x, w = np.polynomial.legendre.leggauss(n)
    # 變換積分區間到 [a, b]
    t = 0.5 * (x + 1) * (b - a) + a
    return 0.5 * (b - a) * np.sum(w * f(t))

# 設定積分區間
a, b = 1, 1.5

# 計算 Gaussian Quadrature 近似值
I_gauss_3 = gauss_quadrature(a, b, 3)
I_gauss_4 = gauss_quadrature(a, b, 4)
```

input

```
Gaussian Quadrature (n=3): 0.19225938
Gaussian Quadrature (n=4): 0.19225936
Exact Integral Value: 0.19225936

...Program finished with exit code 0
```

3.



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

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

# 積分上下限
def y_lower(x):
    return np.sin(x)

def y_upper(x):
    return np.cos(x)

x0, x1 = 0, np.pi / 4

# (a) Simpson's Rule 實作
def simpsons_double(f, x0, x1, nx, ny):
    x = np.linspace(x0, x1, nx + 1)
```

input

```
Simpson's Rule     (n=4, m=4): 0.51198754
Gaussian Quadrature (n=3, m=3): 0.51186554
Exact Value                   : 0.51184464

Simpson 誤差: 1.43e-04
Gaussian 誤差: 2.09e-05
```

4.

```
import numpy as np
import scipy.integrate as spi

def simpsons_rule(f, a, b, n):
    if n % 2 == 1:
        raise ValueError("n must be even for Simpson's rule")

    h = (b - a) / n
    x = np.linspace(a, b, n + 1)
    y = f(x)

    integral = (h / 3) * (y[0] + 4 * sum(y[1:n:2]) + 2 * sum(y[2:n-1:2]) + y[n])
    return integral

def f1(x):
    return x**(-1/4) * np.sin(x)

def f2(x):
    return x**(-4) * np.sin(x)
```

input

```
Approximate integral for (a): 6.94708410137921
Approximate integral for (b): 0.27465025002428546


...Program finished with exit code 0
Press ENTER to exit console.
```