```python
# part a
import sympy as sp
x, y = sp.symbols('x y')
# Define a function
f = sp.exp(x)*sp.sin(y) + y**3
# Differentiate f with respect to x
df_dx = sp.diff(f, x)
# Differentiate f with respect to y
df_dy = sp.diff(f, y)
print("Partial derivative with respect to x:")
print(df_dx)

print("Partial derivative with respect to y:")
print(df_dy)
```

```
Partial derivative with respect to x:
exp(x)*sin(y)
Partial derivative with respect to y:
3*y**2 + exp(x)*cos(y)
```


```python
# part b
import sympy as sp
x, y = sp.symbols('x y')
# Define the second function
f2 = x**2 * y + x * y**2
```


```python
# part b
import sympy as sp
x, y = sp.symbols('x y')
# Define the second function
f2 = x**2 * y + x * y**2
# Differentiate f with respect to x
df2_dx = sp.diff(f2, x)
# Differentiate f with respect to y
df2_dy = sp.diff(f2, y)

# Sub in x and y-values of the point
x_value = 1
y_value = -1
df2_dxsubbed = df2_dx.subs({x: x_value, y: y_value})
df2_dysubbed = df2_dy.subs({x: x_value, y: y_value})
print("Partial derivative with respect to x:")
print(df2_dx)

print("Partial derivative with respect to y:")
print(df2_dy)

gradient = sp.sqrt(df2_dx**2 + df2_dy**2)
print("The gradient vector is", gradient)

gradient = sp.sqrt(df2_dxsubbed**2 + df2_dysubbed**2)
print("The magnitude of the gradient vector at (1,-1) is", gradient)
```

```
print("Partial derivative with respect to y:")
print(df2_dy)

gradient = sp.sqrt(df2_dx**2 + df2_dy**2)
print("The gradient vector is", gradient)

gradient = sp.sqrt(df2_dxsubbed**2 + df2_dysubbed**2)
print("The magnitude of the gradient vector at (1,-1) is", gradient)
```

```
Partial derivative with respect to x:
2*x*y + y**2
Partial derivative with respect to y:
x**2 + 2*x*y
The gradient vector is sqrt((x**2 + 2*x*y)**2 + (2*x*y + y**2)**2)
The magnitude of the gradient vector at (1,-1) is sqrt(2)
```

```
#part c
import sympy as sp
x, y = sp.symbols ('x y')
# Define third function
f3 = sp.log(x**2 + y**2)
# Differentiate f with respect to x
df3_dx = sp.diff(f3, x)
# Differentiate f with respect to y
df3_dy = sp.diff(f3, y)
# Second partial derivatives
d2f3_dx2= sp.diff(df3_dx, x)
```

```
#part c
import sympy as sp
x, y = sp.symbols ('x y')
# Define third function
f3 = sp.log(x**2 + y**2)
# Differentiate f with respect to x
df3_dx = sp.diff(f3, x)
# Differentiate f with respect to y
df3_dy = sp.diff(f3, y)
# Second partial derivatives
d2f3_dx2= sp.diff(df3_dx, x)
d2f3_dy2= sp.diff(df3_dy, y)
mixeddf3_dx=sp.diff(df3_dx, y)
mixeddf3_dy=sp.diff(df3_dy, x)
print("Partial derivative with respect to x:")
print(df3_dx)

print("Partial derivative with respect to y:")
print(df3_dy)

print("Second unmixed partial derivative with respect to x:")
print(d2f3_dx2)

print("Second unmixed partial derivative with respect to y:")
print(d2f3_dy2)
```

```
print("Second unmixed partial derivative with respect to x:")
print(d2f3_dx2)

print("Second unmixed partial derivative with respect to y:")
print(d2f3_dy2)

print("Second mixed partial derivatives:")
print(mixeddf3_dy)
print(mixeddf3_dx)

print("The mixed partial derivatives are symmetrical as a result of Clairaut's theorem. It states that if fxy and fy
```

```
Partial derivative with respect to x:
2*x/(x**2 + y**2)
Partial derivative with respect to y:
2*y/(x**2 + y**2)
Second unmixed partial derivative with respect to x:
-4*x**2/(x**2 + y**2)**2 + 2/(x**2 + y**2)
Second unmixed partial derivative with respect to y:
-4*y**2/(x**2 + y**2)**2 + 2/(x**2 + y**2)
Second mixed partial derivatives:
-4*x*y/(x**2 + y**2)**2
-4*x*y/(x**2 + y**2)**2
```

```
#part d
#How to create a contour plot of a function
```

[2]:

```
#part d
#How to create a contour plot of a function
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt
x, y = sp.symbols ('x y')
j = x**3 - 3*x*y + y**3
j_func = sp.lambdify((x, y), j, 'numpy')
x_vals = np.linspace(-3, 3, 400)
y_vals = np.linspace(-3, 3, 400)
X, Y = np.meshgrid(x_vals, y_vals)
Z = j_func(X, Y)
plt.contourf(X, Y, Z, levels=50, cmap='viridis')
plt.colorbar()
plt.title('Contour plot of $j(x, y) = x^3 - 3xy + y^3$')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.show()
```