In [1]:
```
print("Hello from Python!")
```

```
Hello from Python!
```

In [2]:
```
pip install sympy
```

```
Requirement already satisfied: sympy in /Users/hamed.baghal/opt/ana
conda3/lib/python3.9/site-packages (1.10.1)
Requirement already satisfied: mpmath>=0.19 in /Users/hamed.baghal/
opt/anaconda3/lib/python3.9/site-packages (from sympy) (1.2.1)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
#Question 1 part a

from sympy import symbols, tan, sin, sqrt, diff


x = symbols("x")


expr = tan(x) + sqrt(sin(x))


derivative_expr = diff(expr, x)


print(derivative_expr)
```

```
tan(x)**2 + 1 + cos(x)/(2*sqrt(sin(x)))
```

In [4]:
```
#Question 1 part b

from sympy import symbols, diff

x = symbols("x")


g = 4*x**3 + x**2 + 35


derivative_g = diff(g, x)

print("Derivative of g(x):", derivative_g)
```

```
Derivative of g(x): 12*x**2 + 2*x
```

In [5]:
```
#Question 1 part c

from sympy import symbols, sin, diff

# Define the symbolic variables
x, y = symbols("x y")

# Define the expression h
h = y * sin(x)

# Compute the partial derivative of h with respect to y
g = diff(h, y)
print("g =", g)

# Compute the partial derivative of g with respect to x
result = diff(g, x)
print("Result =", result)
```

```
g = sin(x)
Result = cos(x)
```

In [6]:
```
#Question 2 part a
from sympy import symbols, solve

# Define the symbolic variable
x = symbols("x")

# Define the equation
equation = x**2 + 5*x + 6

# Solve the equation
solutions = solve(equation, x)

# Print the solutions
print("Solutions:", solutions)
```

```
Solutions: [-3, -2]
```

In [7]:
```
#Question 2 part b

from sympy import symbols, solve

# Define the symbolic variable
x = symbols("x")

# Define the equation
equation = x**3 + 2*x

# Solve the equation
solutions = solve(equation, x)

# Print the solutions
print("Solutions:", solutions)
```

```
Solutions: [0, -sqrt(2)*I, sqrt(2)*I]
```

In [8]:
```
from sympy import symbols, solveset, S

# Define the symbolic variable
x = symbols("x")

# Define the equation
equation = x**3 + 2*x

# Solve the equation for real solutions
solutions = solveset(equation, x, domain=S.Reals)

# Print the solutions
print("Real Solutions:", solutions)
```

```
Real Solutions: {0}
```

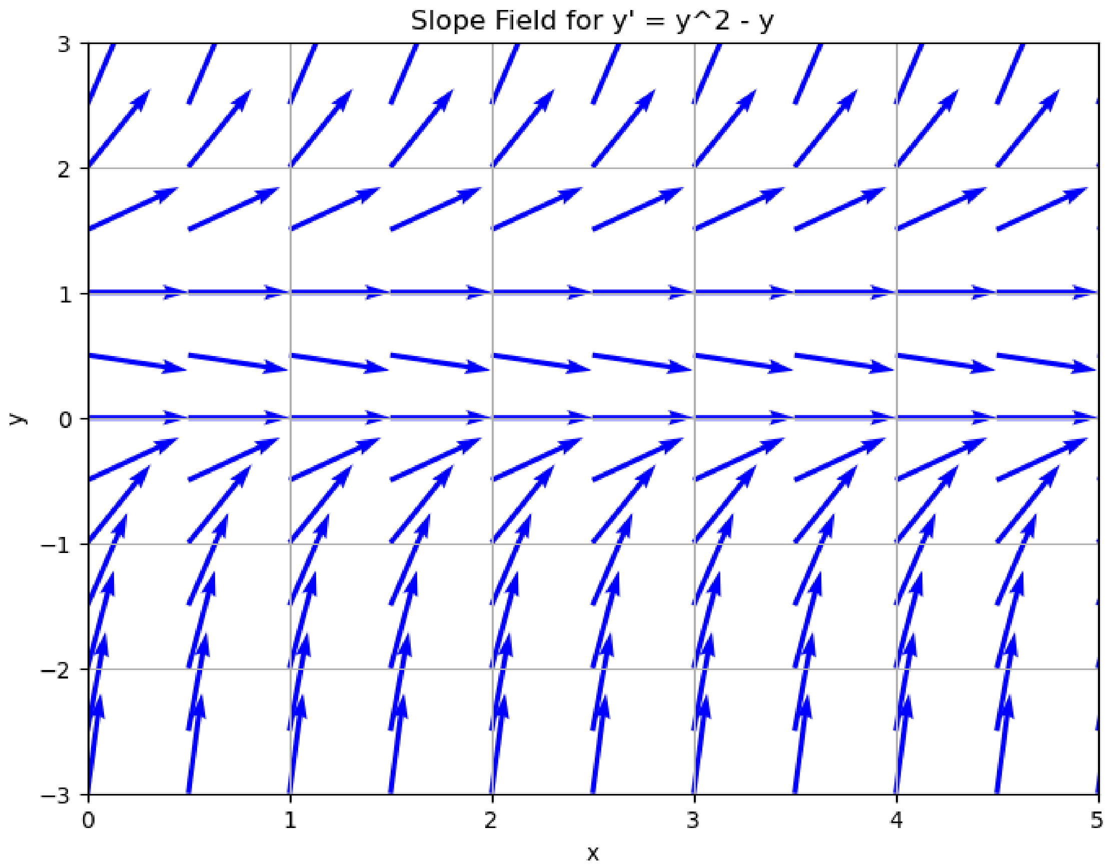
In [9]:
```
from sympy import solve

# Display documentation for the solve function
help(solve)
```

```
Help on function solve in module sympy.solvers.solvers:

solve(f, *symbols, **flags)
    Algebraically solves equations and systems of equations.

    Explanation
    ===========

    Currently supported:
        - polynomial
        - transcendental
        - piecewise combinations of the above
        - systems of linear and polynomial equations
        - systems containing relational expressions

    Examples
    ========

    The output varies according to the input and can be seen by e
xample:
```

In [10]:
```
#Question 3 part a

from sympy import symbols, Function, diff, dsolve

# Define the symbolic variable and function
x = symbols("x")
y = Function("y")(x)

# Define the differential equation
DE = diff(y, x) - 2*x - y

# Solve the differential equation
solution = dsolve(DE, y)

# Print the solution
print("Solution:", solution)
```

```
Solution: Eq(y(x), C1*exp(x) - 2*x - 2)
```

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the function f(x, y)
def f(x, y):
    return y**2 - y

# Define the range for x and y
x_range = np.arange(0, 5.5, 0.5)
y_range = np.arange(-3, 3.5, 0.5)

# Create a grid of points
X, Y = np.meshgrid(x_range, y_range)

# Compute the slopes
U = np.ones_like(X)  # dx is always 1 for slope fields
V = f(X, Y)          # dy/dx = f(x, y)

# Normalize the slopes for better visualization
magnitude = np.sqrt(U**2 + V**2)
U /= magnitude
V /= magnitude

# Plot the slope field
plt.figure(figsize=(8, 6))
plt.quiver(X, Y, U, V, angles="xy", scale=10, color="blue")

# Customize the plot
plt.xlim(0, 5)
plt.ylim(-3, 3)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Slope Field for y' = y^2 - y")
plt.grid()
plt.show()
```



In [2]:
```
from sympy import symbols, Function, diff, dsolve

# Define the symbolic variable and function
x = symbols("x")
y = Function("y")

# Define the differential equation
DE = diff(y(x), x) - (y(x)**2 - y(x))  # Use ** for exponentiation i

# Solve the differential equation
solution = dsolve(DE, y(x))

# Print the solution
print("Solution:", solution)
```

```
Solution: Eq(y(x), C1/(C1 - exp(x)))
```

In [11]:
```
#Question 3 part b

from sympy import symbols, Function, diff, dsolve, tan

# Define the symbolic variable and function
x = symbols("x")
y = Function("y")(x)

# Define the differential equation
DE = diff(y, x) - tan(x + y) + 1

# Solve the differential equation
solution = dsolve(DE, y)

# Print the solution
print("Solution:", solution)
```

```
Solution: [Eq(y(x), -x - asin(C1*exp(x))), Eq(y(x), -x + asin(C1*ex
p(x)) + pi)]
```

In [12]:
```
from sympy import dsolve

# Display documentation for the solve function
help(dsolve)
```

```
Help on function dsolve in module sympy.solvers.ode.ode:

dsolve(eq, func=None, hint='default', simplify=True, ics=None, xi
=None, eta=None, x0=0, n=6, **kwargs)
    Solves any (supported) kind of ordinary differential equatio
n and
    system of ordinary differential equations.

    For single ordinary differential equation
    =========================================

    It is classified under this when number of equation in ``eq`
` is one.
    **Usage**

        ``dsolve(eq, f(x), hint)`` -> Solve ordinary differentia
l equation
        ``eq`` for function ``f(x)``, using method ``hint``.

    **Details**
```

In [13]:
```
#Question 3 part c

from sympy import symbols, Function, diff, dsolve, pi

# Define the symbolic variable and function
x = symbols("x")
y = Function("y")(x)

# Define the differential equation
DE = diff(y, x, x) + 2*diff(y, x) + y

# Define the initial conditions
# y(0) = 3, y'(pi/2) = 2
ics = {y.subs(x, 0): 3, y.subs(x, pi/2): 2}

# Solve the differential equation with initial conditions
solution = dsolve(DE, y, ics=ics)

# Print the solution
print("Solution:", solution)
```

```
Solution: Eq(y(x), (x*(-6 + 4*exp(pi/2))/pi + 3)*exp(-x))
```

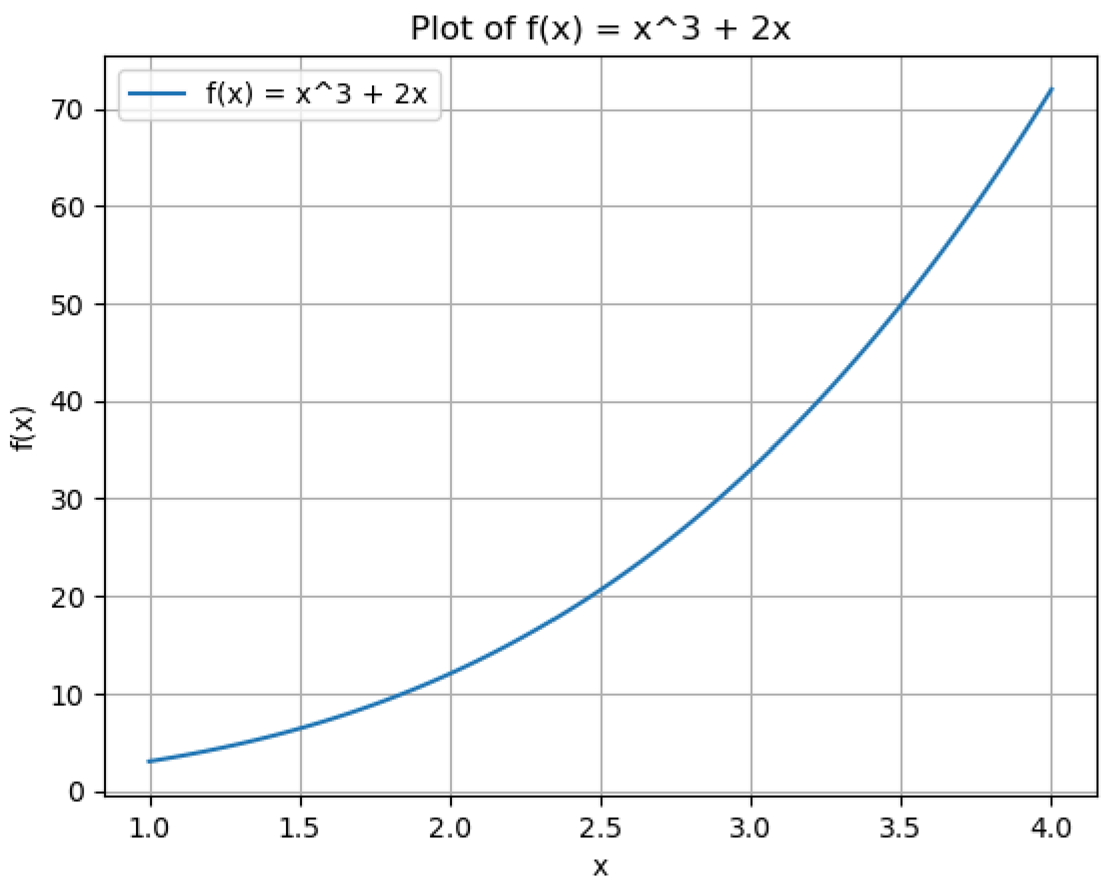
In [14]:
```
#Question 3 part c'

from sympy import symbols, Function, diff, dsolve, pi

# Define the symbolic variable and function
x = symbols("x")
y = Function("y")(x)

# Define the differential equation
DE = diff(y, x, x) + 2*diff(y, x) + y

# Define the initial conditions
# y(0) = 3, y'(pi/2) = 2
ics = {y.subs(x, 0): 3, diff(y, x).subs(x, pi/2): 2}

# Solve the differential equation with initial conditions
solution = dsolve(DE, y, ics=ics)

# Print the solution
print("Solution:", solution)
```

```
Solution: Eq(y(x), (x*(-4*exp(pi) - 6*exp(pi/2))/(-2*exp(pi/2) + pi
*exp(pi/2)) + 3)*exp(-x))
```

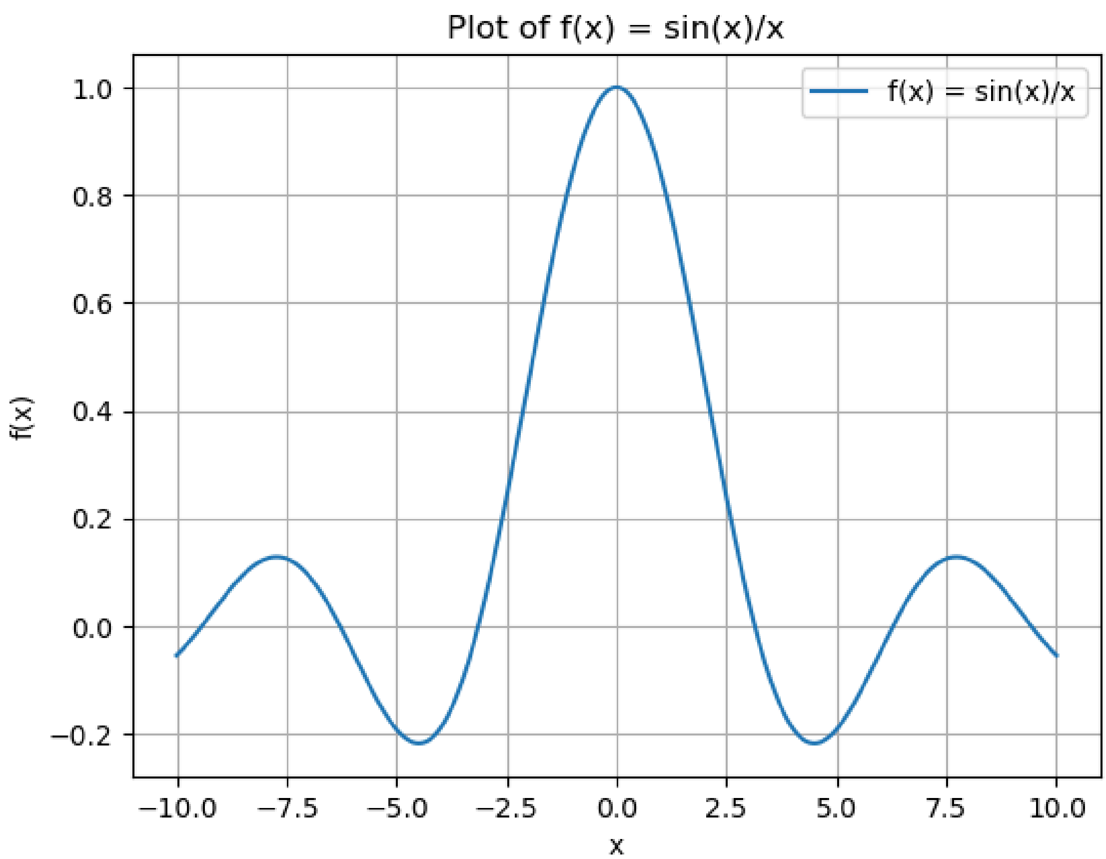
In [15]:
```
pip install matplotlib numpy
```

```
Requirement already satisfied: matplotlib in /Users/hamed.baghal/op
t/anaconda3/lib/python3.9/site-packages (3.5.2)
Requirement already satisfied: numpy in /Users/hamed.baghal/opt/ana
conda3/lib/python3.9/site-packages (1.21.5)
Requirement already satisfied: fonttools>=4.22.0 in /Users/hamed.ba
ghal/opt/anaconda3/lib/python3.9/site-packages (from matplotlib)
(4.25.0)
Requirement already satisfied: python-dateutil>=2.7 in /Users/hame
d.baghal/opt/anaconda3/lib/python3.9/site-packages (from matplotli
b) (2.8.2)
Requirement already satisfied: cycler>=0.10 in /Users/hamed.baghal/
opt/anaconda3/lib/python3.9/site-packages (from matplotlib) (0.11.
0)
Requirement already satisfied: kiwisolver>=1.0.1 in /Users/hamed.ba
ghal/opt/anaconda3/lib/python3.9/site-packages (from matplotlib)
(1.4.2)
Requirement already satisfied: pillow>=6.2.0 in /Users/hamed.bagha
l/opt/anaconda3/lib/python3.9/site-packages (from matplotlib) (9.2.
0)
Requirement already satisfied: pyparsing>=2.2.1 in /Users/hamed.bag
hal/opt/anaconda3/lib/python3.9/site-packages (from matplotlib) (3.
0.9)
Requirement already satisfied: packaging>=20.0 in /Users/hamed.bagh
al/opt/anaconda3/lib/python3.9/site-packages (from matplotlib) (21.
3)
Requirement already satisfied: six>=1.5 in /Users/hamed.baghal/opt/
anaconda3/lib/python3.9/site-packages (from python-dateutil>=2.7->m
atplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [16]:
```
#Question 4 part a
import matplotlib.pyplot as plt
import numpy as np

# Define the function
def f(x):
    return x**3 + 2*x

# Define the range for x
x_values = np.linspace(1, 4, 100)  # 100 points between 1 and 4

# Compute the corresponding y values
y_values = f(x_values)

# Plot the function
plt.plot(x_values, y_values, label="f(x) = x^3 + 2x")

# Add labels and title
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Plot of f(x) = x^3 + 2x")

# Add a legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



In [18]:
```
#Question 4 part a

import matplotlib.pyplot as plt
import numpy as np

# Define the function
def f(x):
    # Handle division by zero at x = 0
    return np.where(x != 0, np.sin(x) / x, 1.0)

# Define the range for x
x_values = np.linspace(-10, 10, 1000)  # 1000 points between -10 and
# Compute the corresponding y values
y_values = f(x_values)

# Plot the function
plt.plot(x_values, y_values, label="f(x) = sin(x)/x")

# Add labels and title
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Plot of f(x) = sin(x)/x")

# Add a legend
plt.legend()

# Show the plot
plt.grid(True)
plt.show()
```



In [20]:
```
import numpy as np

# Define the function
def f(x):
    return np.sin(x) / x

# Evaluate the function at x = 1
result = f(1)

# Print the result
print("f(1) =", result)
```

```
f(1) = 0.8414709848078965
```

In [22]:
```
f(0)
```

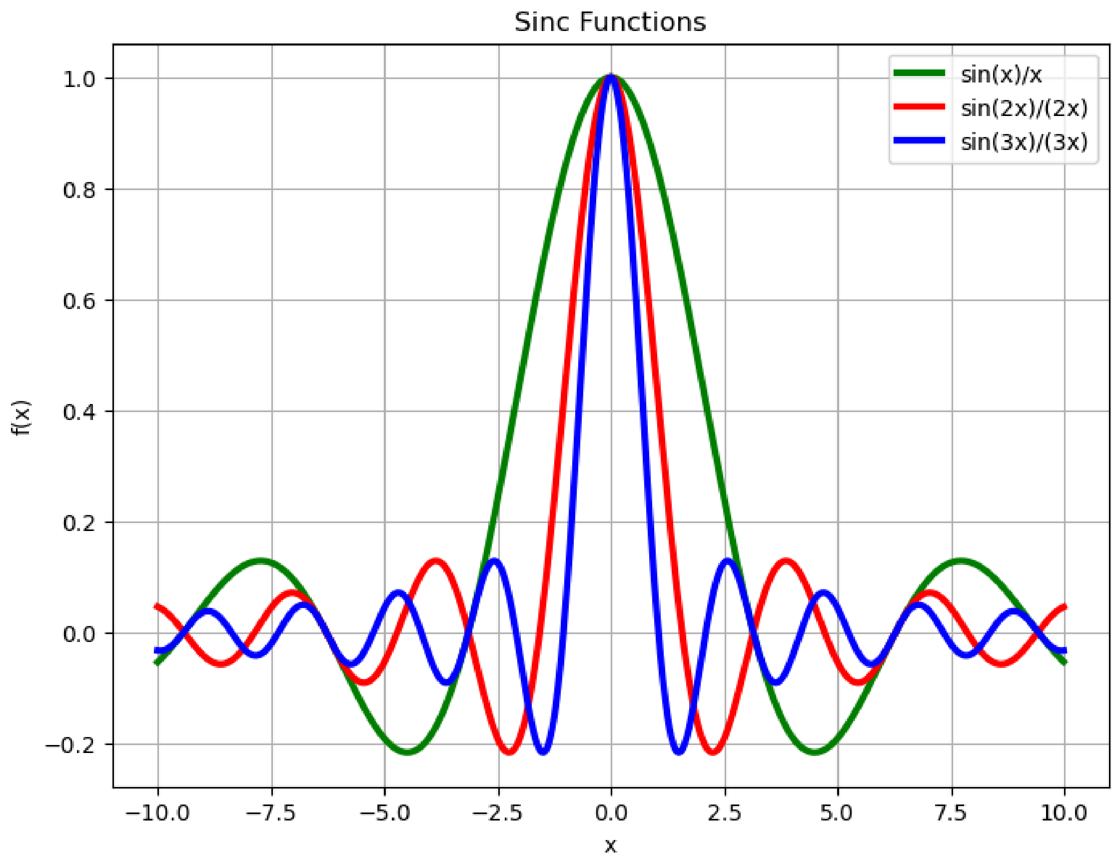
```
/var/folders/q4/79f1wcfj26x062qr_grsw2gw0000gn/T/ipykernel_11284/37
87353187.py:5: RuntimeWarning: invalid value encountered in double_
scalars
  return np.sin(x) / x
```

Out[22]:
```
nan
```

In [23]:
```
from sympy import symbols, Function, diff, dsolve

# Define the symbolic variable and function
x = symbols("x")
y = Function("y")(x)

# Define the differential equation
DE = diff(y, x, x) - diff(y, x) + y

# Solve the differential equation
solution = dsolve(DE, y)

# Print the solution
print("Solution:", solution)
```

```
Solution: Eq(y(x), (C1*sin(sqrt(3)*x/2) + C2*cos(sqrt(3)*x/2))*exp
(x/2))
```

In [25]:
```
import numpy as np
import matplotlib.pyplot as plt

def sinc(n, x):
    return np.where(x == 0, 1.0, np.sin(n * x) / (n * x))

# Define the x range
x = np.arange(-10, 10, 0.01)

# Compute the function values
f0 = sinc(1, x)
f1 = sinc(2, x)
f2 = sinc(3, x)

# Plot the functions
plt.figure(figsize=(8, 6))
plt.plot(x, f0, label="sin(x)/x", linewidth=3, color='green')
plt.plot(x, f1, label="sin(2x)/(2x)", linewidth=3, color='red')
plt.plot(x, f2, label="sin(3x)/(3x)", linewidth=3, color='blue')

# Customize the plot
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Sinc Functions")
plt.legend(loc="upper right")
plt.grid(True)

# Show the plot
plt.show()
```



In [26]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Display documentation for the solve function
help(plt)
```

```
Help on module matplotlib.pyplot in matplotlib:

NAME
    matplotlib.pyplot

DESCRIPTION
    `matplotlib.pyplot` is a state-based interface to matplotlib.
It provides
    an implicit,  MATLAB-like, way of plotting.  It also opens fi
gures on your
    screen, and acts as the figure GUI manager.

    pyplot is mainly intended for interactive plots and simple ca
ses of
    programmatic plot generation::

        import numpy as np
        import matplotlib.pyplot as plt

        x = np.arange(0, 5, 0.1)
```

In [27]:
```python
#Question 6 part a
s=0
for i in range (100):
    if i%2==1:
        s=s+i
print(s)
```

```
2500
```

In [35]:
```python
#BTW what is the imaginary number
1j**2
```

Out[35]:
```
(-1+0j)
```

In [44]:
```python
import math

math.sqrt(math.pi)
```

Out[44]:
```
1.7724538509055159
```

In [46]:
```
#Sometimes "desolve" can't help to solve a problem. The following ex

from sympy import symbols, Function, Eq, diff, sin
from sympy.solvers.ode import dsolve

# Define variables
x = symbols('x')
y = Function('y')(x)

# Define the differential equation
DE = Eq(diff(y, x) - (sin(x) * y / x) + x, 0)

# Solve the differential equation
solution = dsolve(DE, y)
print(solution)
```

```
Eq(Integral((x**2 - y(x)*sin(x))*exp(-Si(x))/x, x), C1)
```

In [48]:
```
# Creating a list
p=list()
p.append(2)
p
```

Out[48]:
```
[2]
```

In [49]:
```
p.append(7)
p
```

Out[49]:
```
[2, 7]
```

In [52]:
```
# Creating a list
pts=[2, 3]
p.append(pts)
p
```

Out[52]:
```
[2, 7, [2, 3]]
```

In [53]:
```
import numpy as np

# Define a vector
pts = np.array([2, 3])

print(pts)
```

```
[2 3]
```

In [54]:
```
#Creating a Function
def f(a,b):
    f=a+b
    return f
f(2,3)
```

Out[54]:
```
5
```

In [55]:
```
def allin(n):
    v=[]
    for i in range(n):
        v.append(i)
    return v
allin(9)
```

Out[55]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [57]:
```
def even(n):
    v=[]
    for i in range(n):
        if i%2==1:
            v.append(i)
    return v
even(10)
```

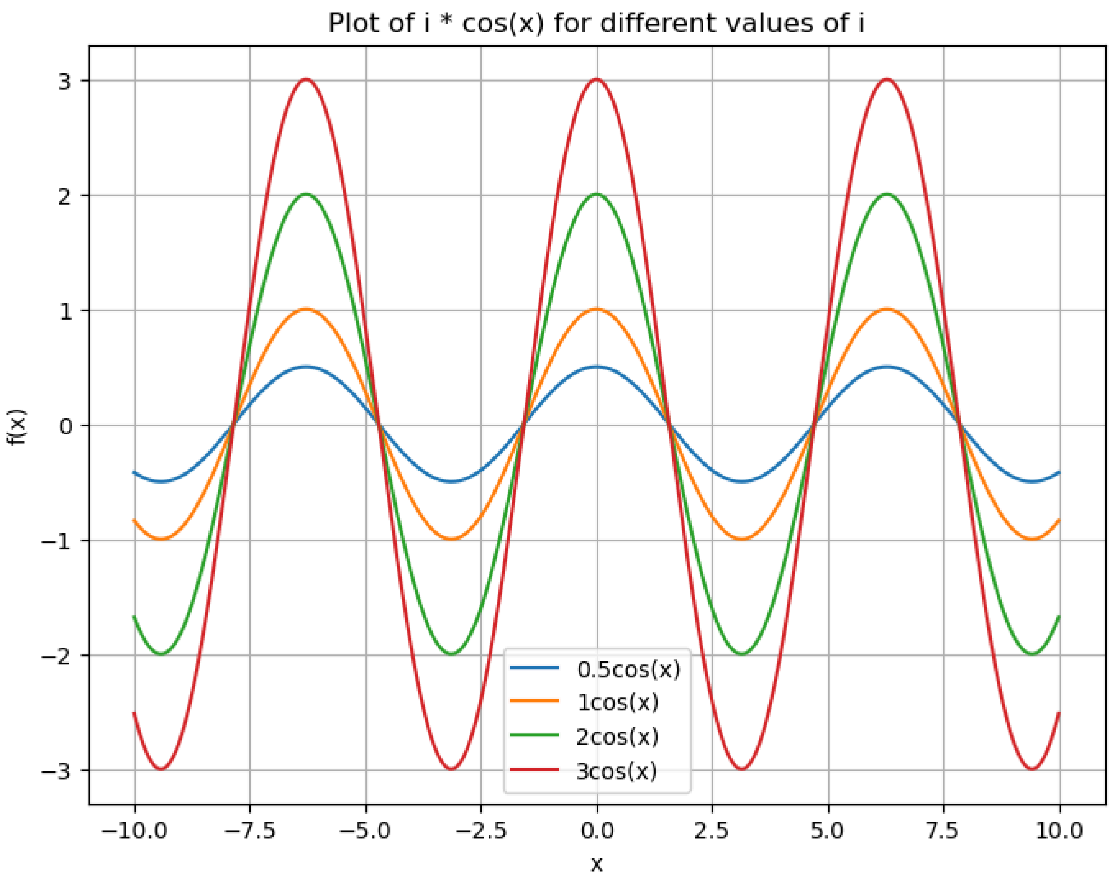
Out[57]:
```
[1, 3, 5, 7, 9]
```

In [66]:
```
from sympy import sin, Symbol, N

def f(x):
    if x == 0:
        return 1
    else:
        return N(sin(x) / x)

# Example usage:
x = Symbol('x')

print(f(0))
f(2)
```

```
1
```

Out[66]: 0.454648713412841

In [67]:
```
import numpy as np
import matplotlib.pyplot as plt

def f(i):
    x = np.linspace(-10, 10, 1000)  # Define x range
    y = i * np.cos(x)  # Compute function values
    plt.plot(x, y, label=f"{i}cos(x)")  # Plot the function

# Plot for different values of i
plt.figure(figsize=(8, 6))
f(0.5)
f(1)
f(2)
f(3)

# Add labels, title, legend, and grid
plt.xlabel("x")
plt.ylabel("f(x)")
plt.title("Plot of i * cos(x) for different values of i")
plt.legend()
plt.grid()

# Show the plot
plt.show()
```



In [ ]: