In [8]:
```
"""
Arjun Srivastava
arj1
AMATH 301 B
"""

import numpy as np
import math
import matplotlib.pyplot as plt

%matplotlib inline
```

In [31]:
```
# Problem 1

x, y, z = 3.1416, 3.141592653589793238462643383279502884197, np.pi

# a)

print('a)', x, y, z, sep='\n')

# b)

print('b)', x - z, y - z, sep='\n')

# c)

print('c)', z - 3.14159265358979, sep='\n')


"""
a) x is less precise than y and z. y and z are not the same value on declarati
on, but they look the same when printed
    due to Python's 16 digit limit for floats.

b) x - z is not 0, but it is very close (7.346410206832132e-06). y - z is actu
ally 0. This is surprising to me,
    but it makes me think that y's 16 digits are the most accurate representat
ion of pi in Python, which is equal
    to math.pi or np.pi

c) It seems that the first 16 digits of pi are saved by np.pi. I just tried to
subtract the first 15 digits of pi
    from z, and I got a non-zero value afterwards. I think this is a Python-sp
ecific problem, since the language is choosing
    to save memory by only saving a certain number of digits.
"""
```

```
a)
3.1416
3.141592653589793
3.141592653589793
b)
7.346410206832132e-06
0.0
c)
3.1086244689504383e-15
```

Out[31]:
```
"\na) x is less precise than y and z. y and z are not the same value on decla
ration, but they look the same when printed\n    due to Python's 16 digit lim
it for floats.\n\nb) x - z is not 0, but it is very close (7.346410206832132e
-06). y - z is actually 0. This is surprising to me,\n    but it makes me thi
nk that y's 16 digits are the most accurate representation of pi in Python, w
hich is equal\n    to math.pi or np.pi\n    \nc) It seems that the first 16 d
igits of pi are saved by np.pi. I just tried to subtract the first 15 digits
of pi\n    from z, and I got a non-zero value afterwards. I think this is a P
ython-specific problem, since the language is choosing\n    to save memory by
only saving a certain number of digits.\n"
```

In [6]:

```
# Problem 2

fig, ax = plt.subplots(1)
x = np.arange(-2*np.pi,2*np.pi,0.1)
y = np.sin(x)
ax.plot(x, y)

# Function to compute nth term of Taylor Series for sin()
def taylor_sin(x, n):
    value = 0
    for i in range(1, n):
        coef = (-1)**(i+1)
        num = x**((2*i)-1)
        denom = math.factorial((2*i)-1)
        value += coef * (num / denom)
    return value

# Plots each Taylor approximation
for i in range(1,5):
    taylor = [taylor_sin(val, i) for val in x]
    ax.plot(x, taylor)

# Set limits
ax.set_ylim([-7,7])

# Add legend
legend = ['sin function']
for i in range(1, 5):
    legend.append(f'Taylor Series: {i}th Term')

ax.set_title('Taylor Approximations of Sine')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.legend(legend, loc=3, prop={'size': 7})
plt.show()
```