Part B

In [2]:
```
def int_to_binary(number):
    if number == 0:
        return "0"
    bitstring = []
    while number > 0:
        remainder = number % 2
        bitstring.insert(0, remainder)
        number = number // 2
    return bitstring

# Example usage
number = 100
binary_string = int_to_binary(number)
print(binary_string)
```

```
[1, 1, 0, 0, 1, 0, 0]
```

Once you understand how this works you can use Bin or even Format

In [5]:
```
bin(100)
```

Out[5]:
```
'0b1100100'
```

Now the opposite:

In [7]:
```
def binary_to_int(bitstring):
    result = 0
    length = len(bitstring)
    for i in range(length):
        digit = int(binary_string[i])
        power = length - i - 1
        result += digit * (2 ** power)
    return result

# Test
binary_string = [1, 1, 0, 0, 1, 0, 0]
result = binary_to_int(binary_string)
print(result)
```

```
100
```

Using now the Python Library

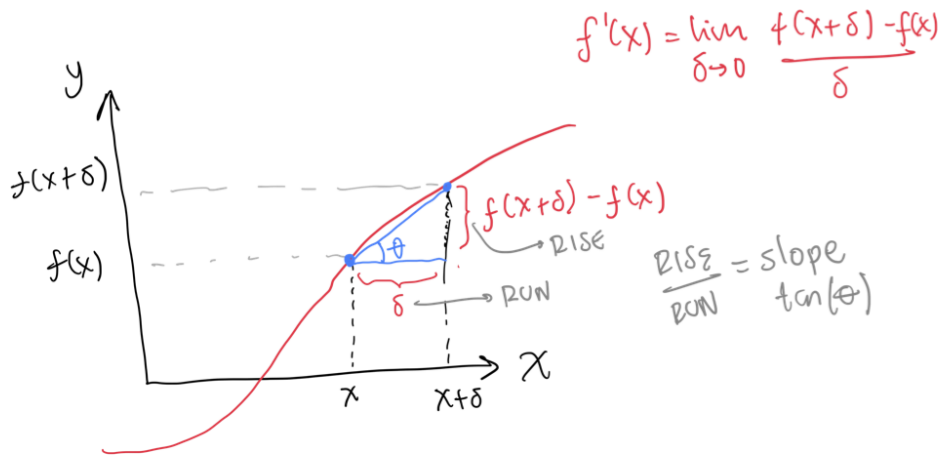
In [9]:
```
int('0b1100100',base=2)
```

Out[9]:
```
100
```

Graphical interpretation of the derivative

In [16]:
```
from IPython.display import Image
Image("graphic.png", width=500, height=500)
```

Out[16]:



Computation using the definition

Now for the derivative using the definition...

$$f'(x) = \lim_{\delta \to 0} \frac{f(x + \delta x - f(x))}{\delta x}$$

Since $f(x) = 5x^2$, then..

$$f'(x) = \lim_{\delta \to 0} \frac{5(x + \delta)^2 - 5x^2}{\delta}$$

$$f'(x) = \lim_{\delta \to 0} \frac{5x^2 + 10x\delta + 5\delta^2 - 5x^2}{\delta}$$

$$f'(x) = \lim_{\delta \to 0} \frac{10x\delta + 5\delta^2}{\delta}$$

$$f'(x) = \lim_{\delta \to 0} 10x + 5\delta$$

$$f'(x) = 10x$$

Should be the same with the formula you learnt before...

Approximation via graphical intuition

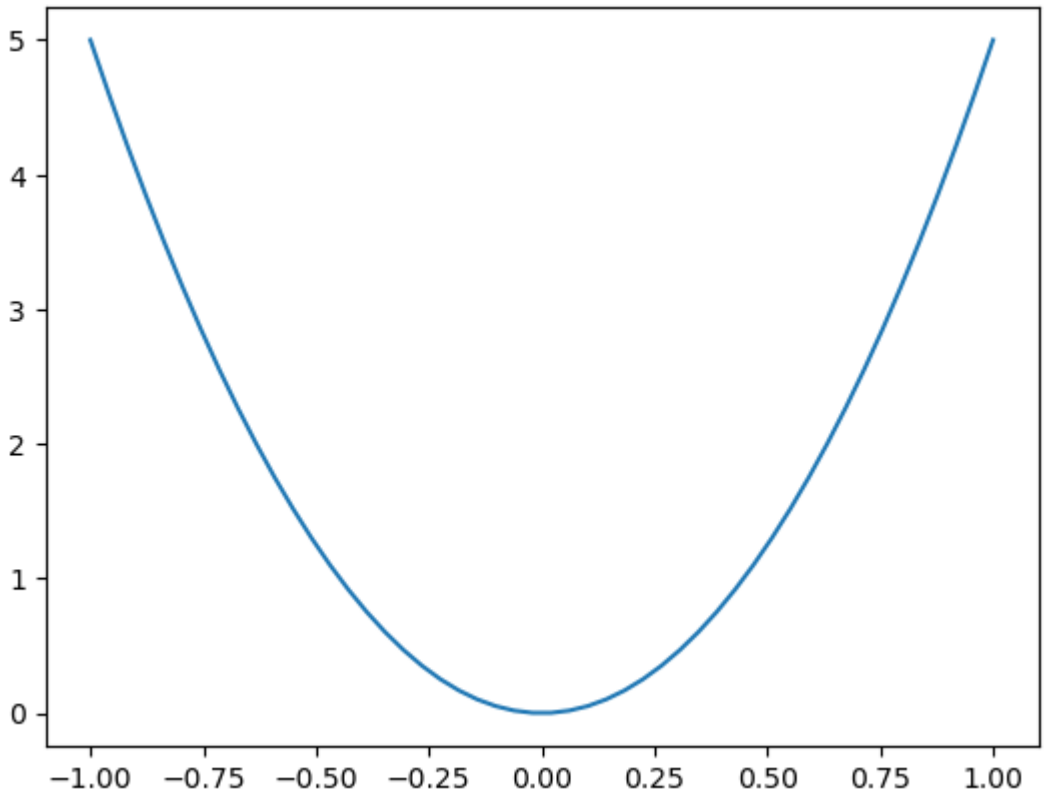
Let us now use the graphical intuition to approximate derivatives of arbitrary functions, visualising the results with plots

In [23]:
```
import numpy as np
import matplotlib.pyplot as plt

def function(x):
    return 5*x*x
```

In [24]:
```
x = np.linspace(-1, 1)
y = [function(x_value) for x_value in x ]
plt.plot(x,y)
```

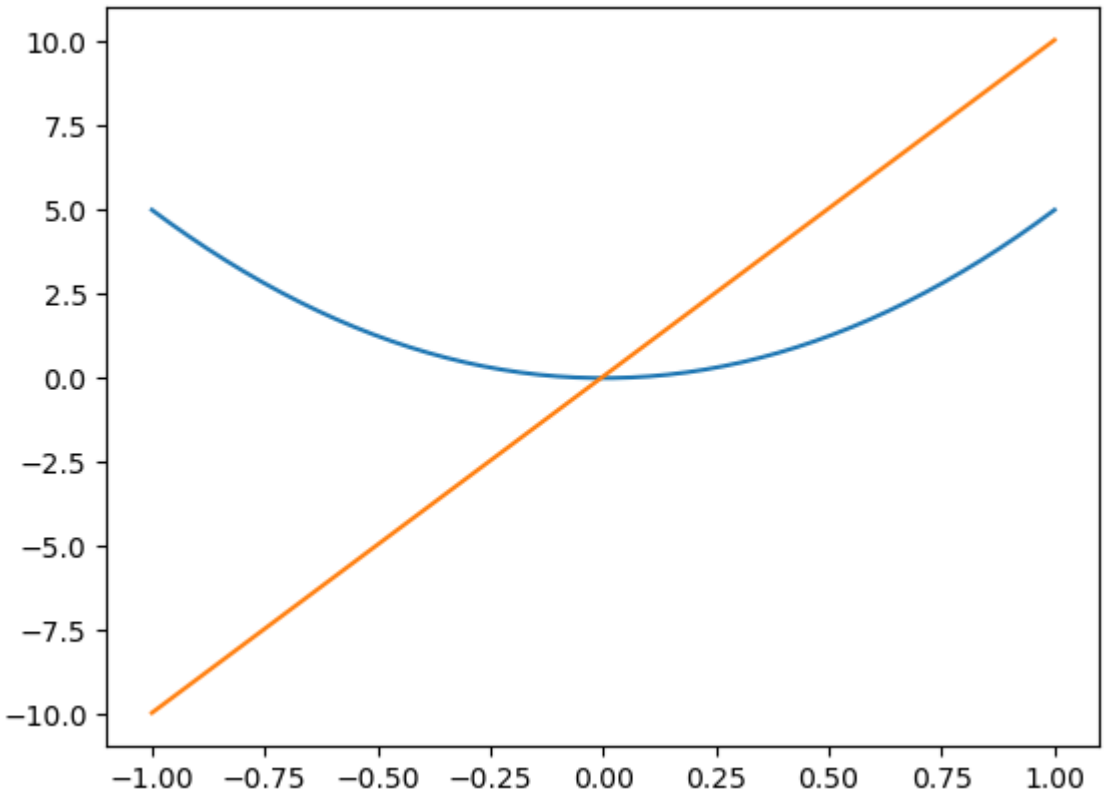
Out[24]:
```
[<matplotlib.lines.Line2D at 0x7fdaea69b700>]
```



In [32]:
```
def derivative_approximation(function, x_values, delta):
    y = [function(x) for x in x_values ]
    y_prime = [(function(x + delta)-function(x))/delta for x in x_values ]
    return y_prime
```

In [33]:
```
x = np.linspace(-1, 1)
y = [function(x_value) for x_value in x ]
y_prime = derivative_approximation(function, x, 0.01)
plt.plot(x,y)
plt.plot(x,y_prime)
```

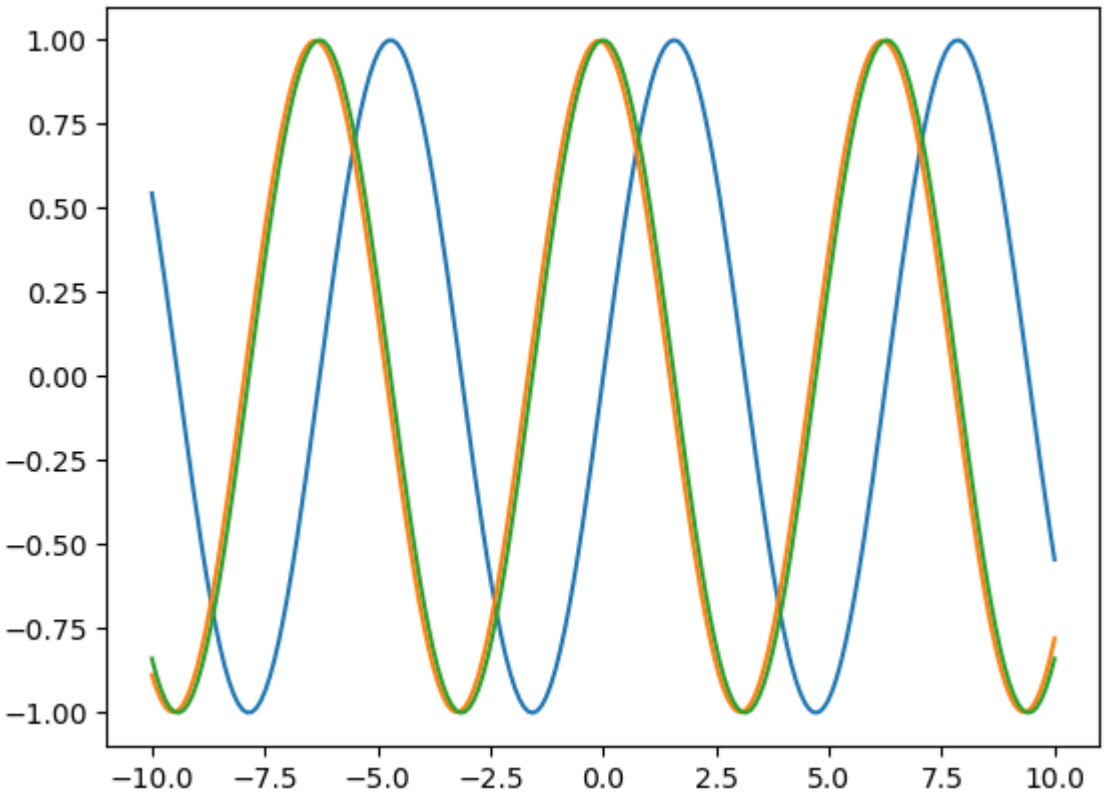
Out[33]:
```
[<matplotlib.lines.Line2D at 0x7fdaea71d760>]
```



Now for something more complicated

In [36]:
```
x = np.linspace(-10, 10, 200)
y = [np.sin(x_value) for x_value in x ]
y_prime = derivative_approximation(np.sin, x, 0.2)
plt.plot(x,y)
plt.plot(x,y_prime)
plt.plot(x, [np.cos(x_value) for x_value in x ])
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x7fdab9e9f6d0>]
```



McLaurin Series

The Maclaurin series for e^x is:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

We can do an approximation on the basis of this

In [37]:
```
def factorial(n):
    if n < 2:
        return 1
    else:
        return n * factorial(n-1)
```

In [41]:
```
def approximation_e(x, number_of_terms):
    summation = 0.0
    for n in range(number_of_terms+1):
        summation = summation + (x**n/factorial(n))
    return summation
```

In [45]:
```
np.exp(5)
```

Out[45]:
```
148.4131591025766
```

In [47]:
```
for i in range(20):
 print(approximation_e(5, i))
```

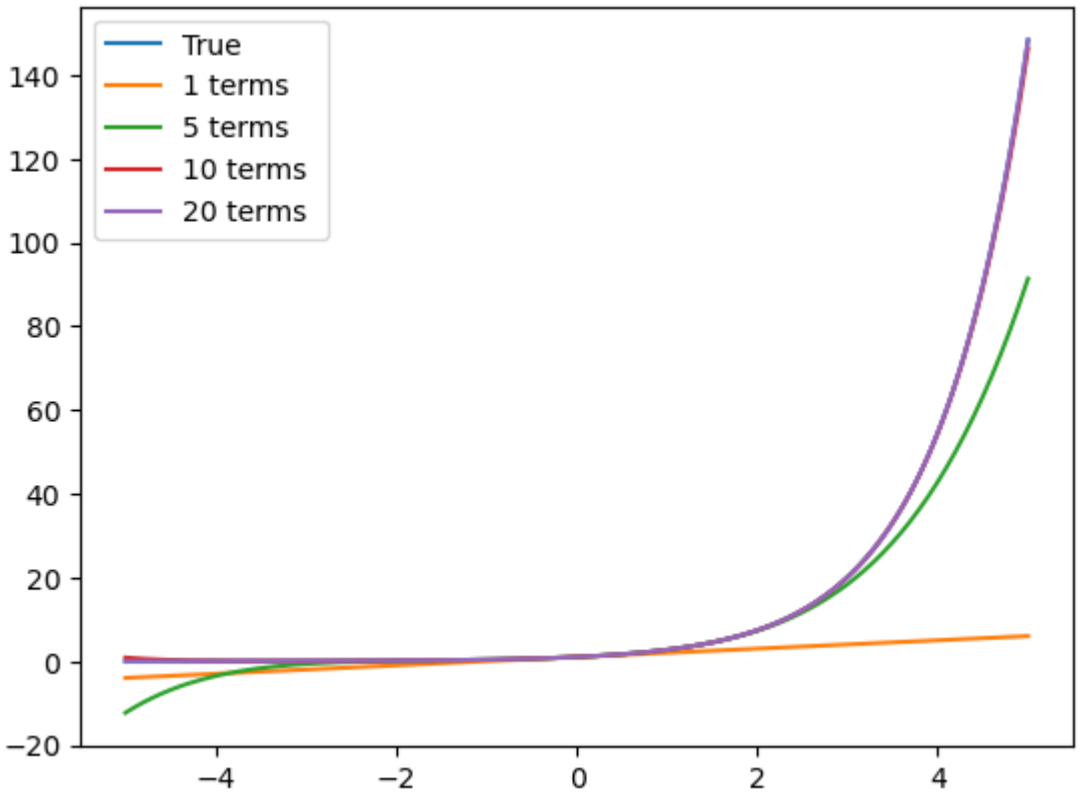
```
1.0
6.0
18.5
39.33333333333333
65.375
91.41666666666667
113.11805555555556
128.61904761904762
138.30716765873015
143.68945656966488
146.38060102513225
147.60384850489015
148.1135349547893
148.3095682047505
148.37958007973663
148.40291737139867
148.41021027504306
148.41235524670316
148.4129510721643
148.4131078683383
```

And now we can visualise it

In [53]:
```
x = np.linspace(-5, 5, 200)
y = [np.exp(x_value) for x_value in x ]
plt.plot(x,y, label="True")
for terms in [1, 5, 10, 20]:
    y_approx = [approximation_e(x_value, terms) for x_value in x]
```

```
    plt.plot(x,y_approx, label="{} terms ".format(terms))
plt.legend()
```

Out[53]: <matplotlib.legend.Legend at 0x7fdad97a4e20>



Challenge: Can you do this for an arbitrary function?

In [ ]: