Doing-math-with-python- / chapter 04 Algebra_&_symbolic_math_with_sympy.ipynb

Open in Colab

chapter 04 Algebra & symbolic math with sympy

In [1]:
```
x = 1
x + x + 1
```

Out[1]: 3

In [2]:
```
from sympy import Symbol
x = Symbol('x')
x
```

Out[2]: x

In [3]:
```
from sympy import symbols
x = Symbol('x')
x + x + 1
```

Out[3]: $2x + 1$

In [4]:
```
a = Symbol('x')
a + a + 1
```

Out[4]: $2x + 1$

In [5]:
```
x = Symbol('x')
y = Symbol('y')
z = Symbol('z')
```

In [6]:
```
from sympy import Symbol
x,y,z = symbols('x,y,z')
```

In [7]:
```
from sympy import Symbol
x = Symbol('x')
y = Symbol('y')
s = x*y + x*y
s
```

Out[7]: $2xy$

In [8]:
```
p = (x+2)*(x +3 )
p
```

Out[8]: $(x+2)(x+3)$

In [9]:

```
#working with the expressions
from sympy import Symbol    # factorizing and expanding
x = Symbol('x')
y = Symbol('y')

from sympy import factor
expr = x**2 - y**2
factor(expr)
```

Out[9]: $(x-y)(x+y)$

In [10]:

```
from sympy import Symbol, factor, expand  # Import expand along with Symbol a

factors = factor(expr)
expand(factors)
```

Out[10]: $x^2 - y^2$

$$x^3 + 3x^2y + 3xy^2 + y^3 = (x+y)^3:$$

In [11]:

```
expr =x**3+3*x**2*y + 3*x*y**2 + y**3
factors = factor(expr)
factors
```

Out[11]: $(x+y)^3$

In [12]:

```
expand(factors)
```

Out[12]: $x^3 + 3x^2y + 3xy^2 + y^3$

In [13]:

```
expr = x + y + x*y
factor(expr)
```

Out[13]: $xy + x + y$

In [14]:

```
expr = x*x + 2*x*y + y*y
expr
```

Out[14]: $x^2 + 2xy + y^2$

In [15]:

```
from sympy import pprint
pprint(expr)
```

```
 2         2
```

```
x  + 2·x·y + y
```

In [16]:

```python
from sympy import init_printing
init_printing(order ='rev -lev ')
print(expr)
```

```
x**2 + 2*x*y + y**2
```

Printing a Series

Consider the following series:

$$x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + \ldots + \frac{x^n}{n}.$$

Let's write a program that will ask a user to input a number, n, and print this series for that number. In the series, x is a symbol and n is an integer input by the program's user. The nth term in this series is given by

$$\frac{x^n}{n}.$$

In [17]:

```python
from sympy import Symbol, pprint, init_printing

def print_series(n):                                   #printing the series

  #intialzing printing system with reverse order
  init_printing(order = 'rev-lex')

  x = Symbol('x')
  series = x
  for i in range(2,n+1):
    series = x
  for i in range(2 , n+1):
    series = series + (x**i)/i
  pprint(series)


if __name__ =='__main__':
  n = input('enter the number of terms you want in the series: ')
  print_series(int(n))
```

```
enter the number of terms you want in the series: 5
     2    3    4    5
    x    x    x    x
x + ── + ── + ── + ──
    2    3    4    5
```

Substituting in Values

In [18]:

```python
x = Symbol('x')
y = Symbol('y')
x*x + x*y + x*y + y*y
```

Out[18]: $y^2 + 2xy + x^2$

In [19]:
```
expr = x*x + x*y + x*y + y*y
res = expr.subs({x:1,y:2})
res
```

Out[19]: 9

In [20]:
```
expr.subs({x:1-y})
```

Out[20]: $(1-y)^2 + 2y(1-y) + y^2$

In [21]:
```
expr_subs = expr.subs({x:1-y})
from sympy import simplify
simplify(expr_subs)
```

Out[21]: 1

In [22]:
```
#calculating the value of a series

from sympy import Symbol, pprint, init_printing

def print_series(n,x_value):

  #intilizing printing system with reverse order
  init_printing(order='rev-lex')

  x = Symbol('x')
  series = x
  for i in range(2,n+1):
    series = series + (x**i)/i

    pprint(series)

    #evalute the series at the x_value
    series_value = series.subs({x:x_value})
    print('value of the series at {0}: {1}'.format(x_value,series_value))

if __name__ == '__main__':
  n = input('enter the number of terms in the series:')
  x_value = input('enter the value of x at which you want to evalute the seri

  print_series(int(n), float(x_value))
```

```
enter the number of terms in the series:5
enter the value of x at which you want to evalute the series :1.2
     2
    x
x + ──
    2
value of the series at 1.2: 1.92000000000000
     2    3
    x    x
x + ── + ──
```

```
  2    3
value of the series at 1.2: 2.49600000000000
     2    3    4
    x    x    x
x + ── + ── + ──
    2    3    4
value of the series at 1.2: 3.01440000000000
     2    3    4    5
    x    x    x    x
x + ── + ── + ── + ──
    2    3    4    5
value of the series at 1.2: 3.51206400000000
```

converting strings to mathematical expressions

In [23]:

```
from sympy import sympify
expr = input('enter an expresion:')
expr = sympify(expr)   # x**2 + 3*x + x**3 + 2*x
2*expr
```

```
enter an expresion: x**2 + 3*x + x**3 + 2*x
```

Out[23]: $10x + 2x^2 + 2x^3$

In [24]:

```
expr = input('enter an expresion:')
```

```
enter an expresion: x**2 + 3*x + x**3 + 2*x
```

In [25]:

```
from sympy import sympify
from sympy.core.sympify import SympifyError
expr = input('enter a mathematical expression:')
try:
  expr = sympify(expr)
except SympifyError:                    #: x**2 + 3*x + x**3 + 2x
  print('invalid input')
else:
  print(expr)
```

```
enter a mathematical expression:: x**2 + 3*x + x**3 + 2x
invalid input
```

expression multiplier

In [26]:

```
from sympy import expand,sympify
from sympy.core.sympify import SympifyError

def product(expr1,expr2):
  prod = expand(expr1*expr2)
  print(prod)


if __name__ == '__main__':
  expr1 = input('enter the first expression:')
  expr2 = input('enter the second expression:')

  try:
    expr1 = sympify(expr1)
```

```
    expr1 = sympify(expr1)
    expr2 = sympify(expr2)                     # x**2 + x*2 + x
  except SympifyError:
    print('invalid input')
  else:                                       # x**3 + x*3 + x

    product(expr1, expr2)
```

```
enter the first expression: x**2 + x*2 + x
enter the second expression: x**3 + x*3 + x
x**5 + 3*x**4 + 4*x**3 + 12*x**2
```

In [27]:

```
if __name__ == '__main__':
  expr1 = input('enter the first expression:')
  expr2 = input('enter the second expression:')

  try:
    expr1 = sympify(expr1)
    expr2 = sympify(expr2)
  except SympifyError:
    print('invalid input')
  else:

    product(expr1, expr2)
    #x*y+x
    # x*x+y
```

```
enter the first expression:x*y+x
enter the second expression: x*x+y
x**3*y + x**3 + x*y**2 + x*y
```

solving equations

In [28]:

```
from sympy import Symbol , solve
x = Symbol('x')
expr = x - 5 - 7
solve(expr)
```

Out[28]: $[12]$

In [29]:

```
from sympy import solve  #solving the quadratic eqs
x = Symbol('x')
expr = x**2 + 5*x + 4
solve(expr, dict = True)
```

Out[29]: $[\{x:-4\},\ \{x:-1\}]$

In [30]:

```
x = Symbol('x')
expr = x**2 + x+1
solve(expr, dict = True)
```

Out[30]: $$\left[\left\{x:-\frac{\sqrt{3}i}{2}-\frac{1}{2}\right\},\ \left\{x:\frac{\sqrt{3}i}{2}-\frac{1}{2}\right\}\right]$$

Solving for One Variable in Terms of Others

Solving for One Variable in Terms of Others

In [31]:

```
x = Symbol('x')
a = Symbol('a')
b = Symbol('b')
c = Symbol('c')
```

In [32]:

```
expr = a*x*x + b*x +c
solve(expr, x, dict = True)
```

Out[32]:

$$\left[\left\{ x : \frac{-\sqrt{b^2 - 4ac} - b}{2a} \right\}, \ \left\{ x : \frac{\sqrt{b^2 - 4ac} - b}{2a} \right\} \right]$$

In [33]:

```
from sympy import Symbol, solve, pprint

s = Symbol('s')
u = Symbol('u')
t = Symbol('t')
a = Symbol('a')

expr = u*t + (1/2)*a*t*t -s
t_expr = solve(expr,t, dict =True)

pprint(t_expr)
```

```
⎡⎧               ___________         ⎫  ⎧                ___________
⎢⎪              ╱      2             ⎪  ⎪               ╱      2
⎢⎨ - 1.4142135623731⋅╲╱  0.5⋅u  + a⋅s  - u⎬  ⎨ 1.4142135623731⋅╲╱  0.5⋅u  + a⋅s  - u⎬⎤
⎢⎪t: ──────────────────────────────────────⎪, ⎪t: ──────────────────────────────────────⎪⎥
⎣⎩                 a                  ⎭  ⎩                 a                  ⎭⎦
```

Solving a System of Linear Equations

Consider the following two equations:

$$2x + 3y = 6$$

$$3x + 2y = 12$$

Say we want to find the pair of values (x, y) that satisfies both the equations. We can use the `solve()` function to find the solution for a system of equations like this one.

In [34]:

```
x = Symbol('x')
y = Symbol('y')
expr1 = 2*x + 3*y -6
expr2 = 3*x + 2*y - 12
solve((expr1,expr2), dict=True)
```

Out[34]: [{ 24 6 }]

$\left[\left\{x : \frac{}{5}, y : -\frac{}{5}\right\}\right]$

In [35]:

```
soln = solve((expr1, expr2), dict = True)
soln = soln[0]
expr1.subs({x:soln[x],y:soln[y]})
```

Out[35]: 0

In [36]:

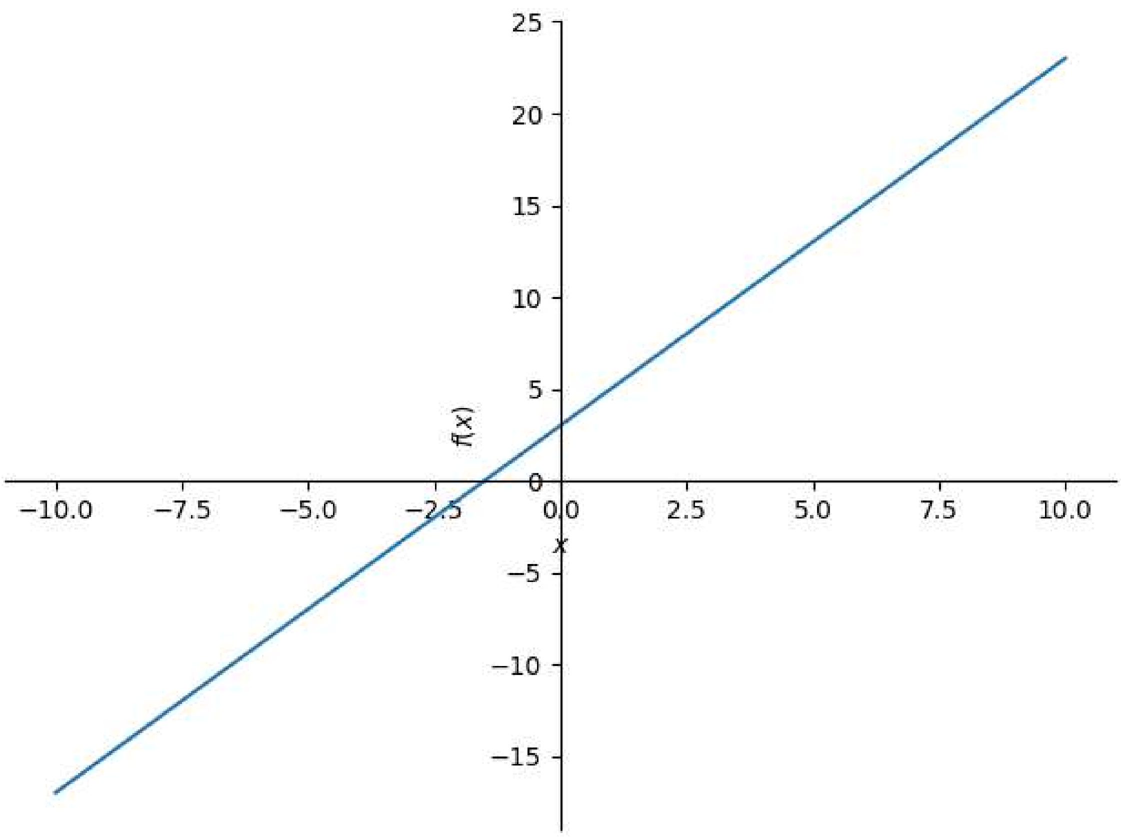
```
expr2.subs({x:soln[x],y:soln[y]})
```

Out[36]: 0

plotting using the sympy

In [37]:

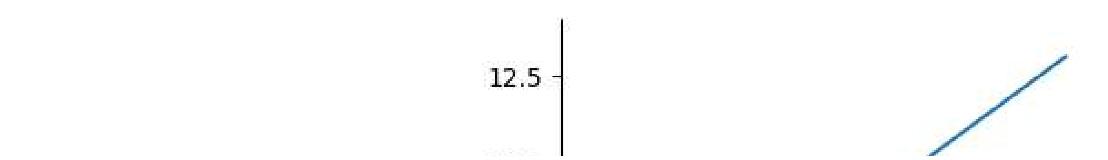
```
from sympy.plotting import plot
from sympy import Symbol
x = Symbol('x')
plot(2*x+3)
```

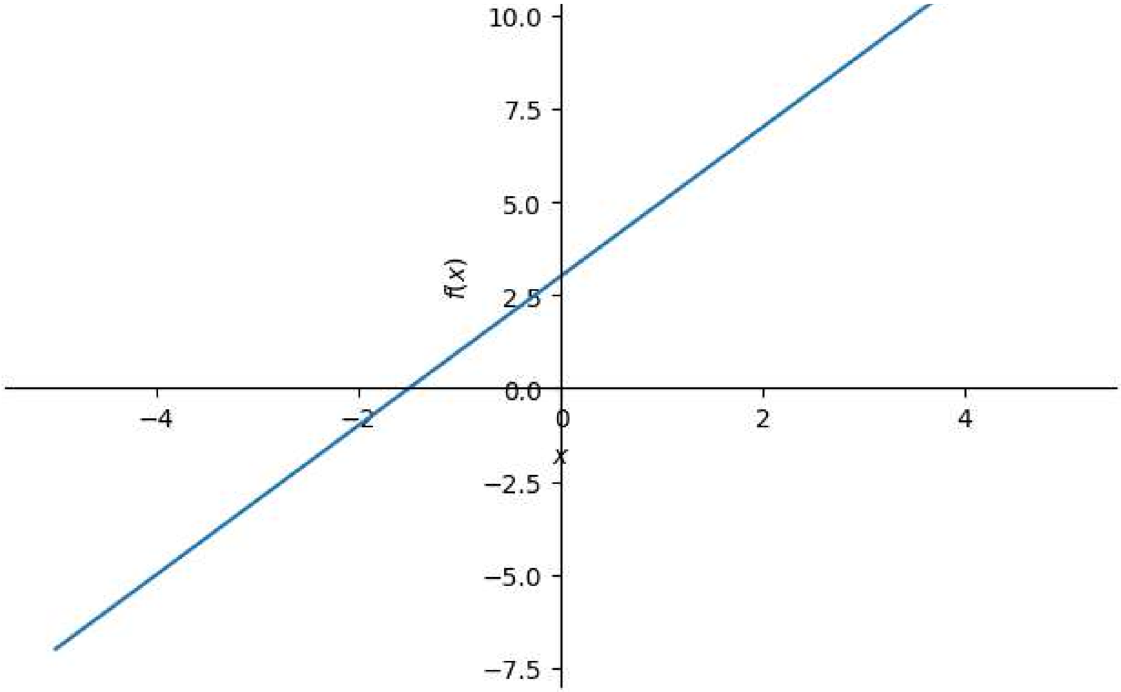


Out[37]: <sympy.plotting.backends.matplotlibbackend.matplotlib.MatplotlibBackend at 0x7bd9e5d8bdf0>

In [38]:

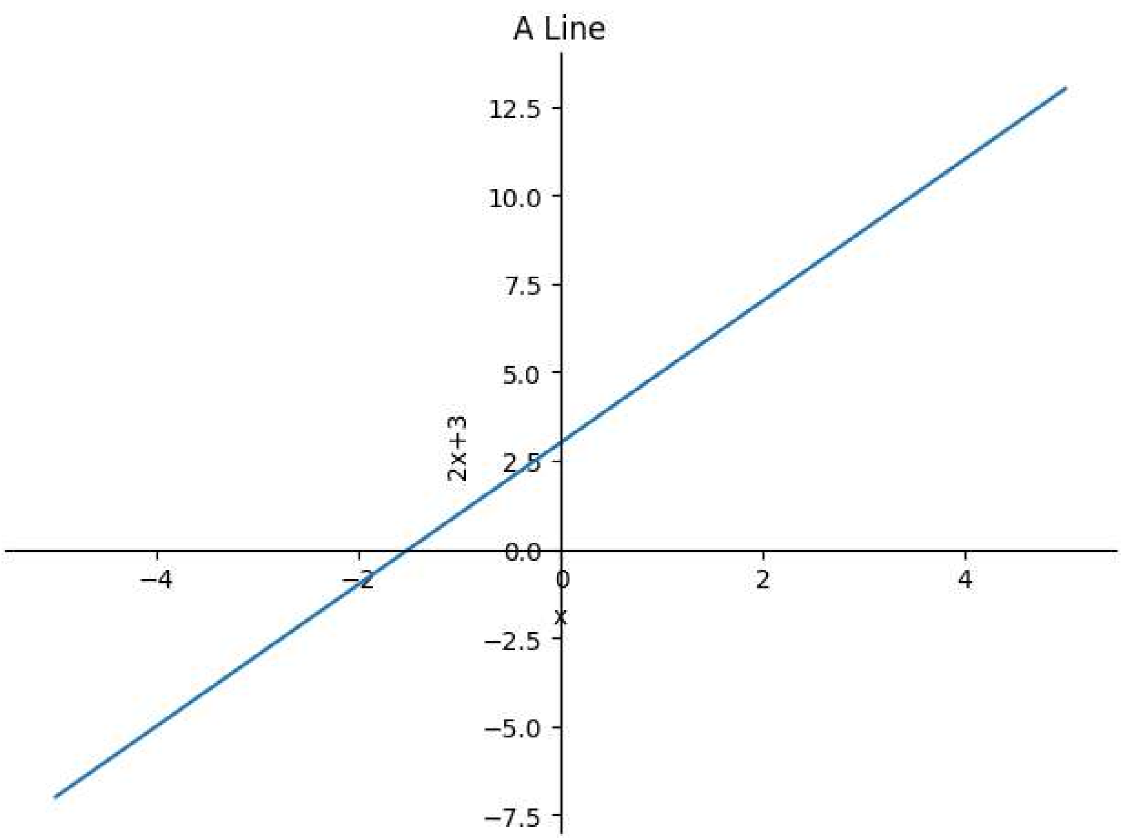
```
plot((2*x + 3), (x, -5,5))
```





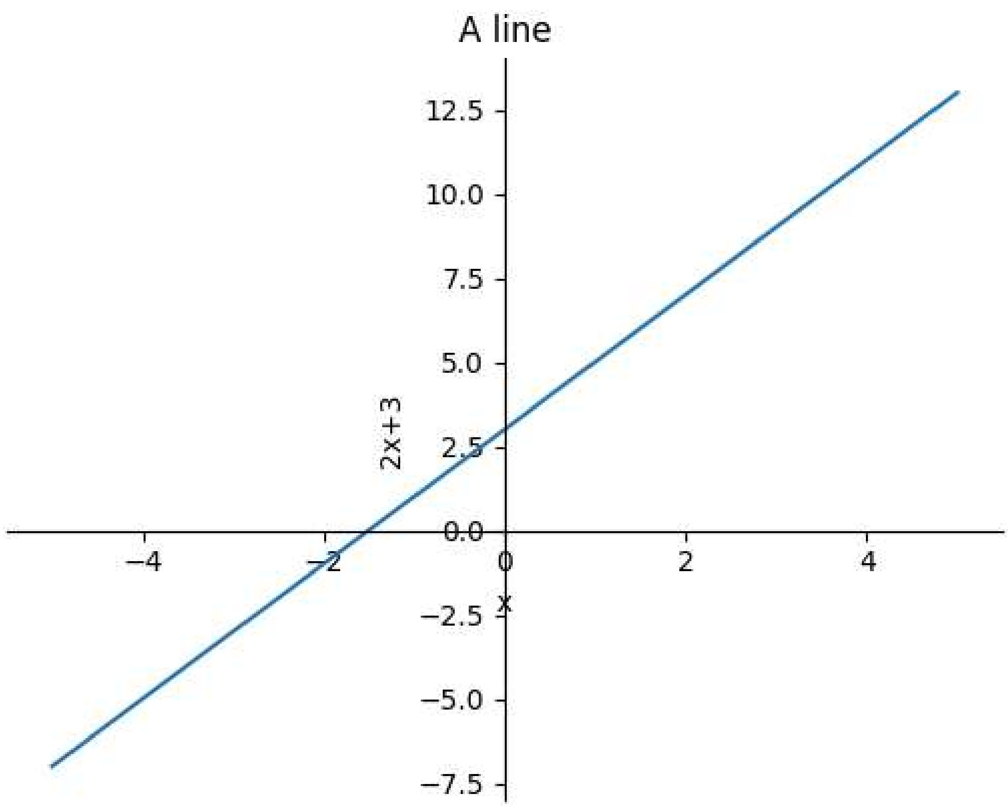
Out[38]: <sympy.plotting.backends.matplotlibbackend.matplotlib.MatplotlibBackend at 0x7bd9e445d840>

In [39]:
```
plot(2*x+3, (x, -5 ,5), title= 'A Line', xlabel='x', ylabel='2x+3')
```



Out[39]: <sympy.plotting.backends.matplotlibbackend.matplotlib.MatplotlibBackend at 0x7bd9e4224cd0>

In [40]:
```
p = plot(2*x + 3, (x, -5 , 5), title ='A line',xlabel='x',ylabel='2x+3', show
p.save('line.png')
```



plotting expression input by the user

In [41]:
```
expr = input('enter an expression:')
expr = sympify(expr)
y = Symbol('y')
solve(expr, y)   #2*x + 3*y -6
```

```
enter an expression:2*x + 3*y -6
```

Out[41]: $\left[2 - \frac{2x}{3}\right]$

In [42]:
```
solutions = solve(expr, y)
expr_y = solutions[0]
expr_y
```

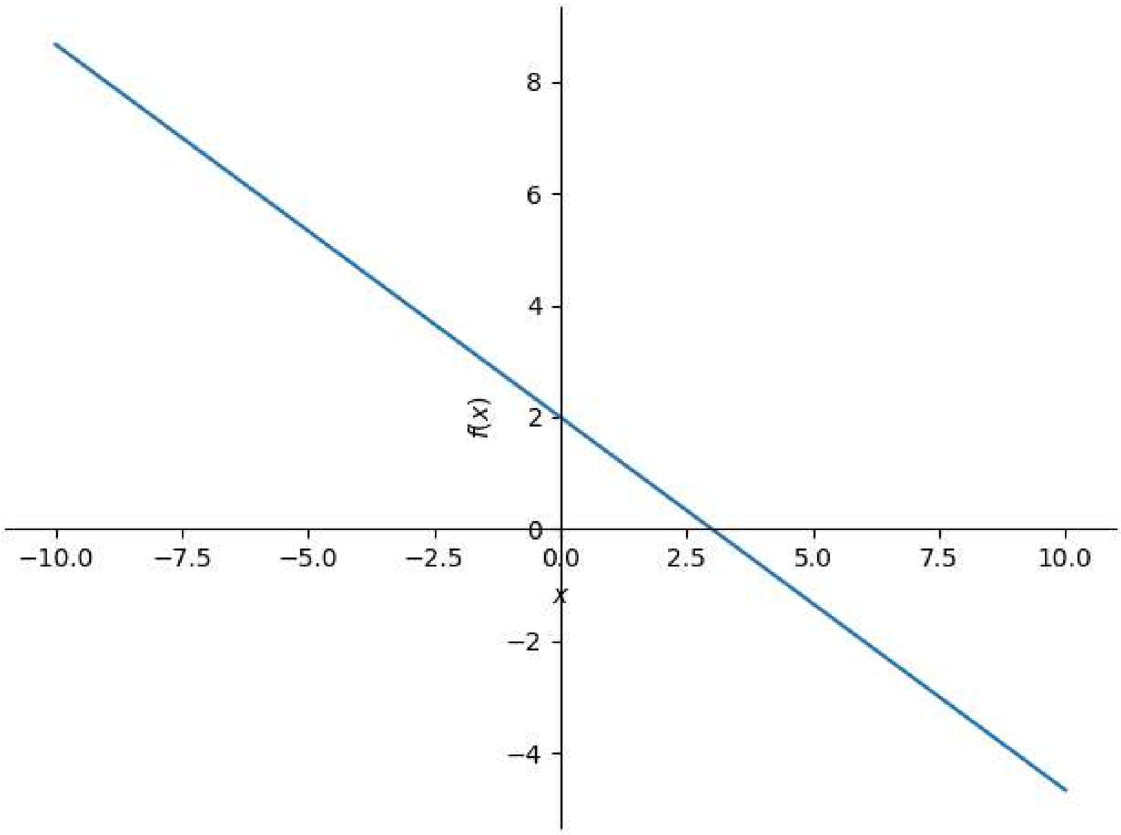
Out[42]: $2 - \frac{2x}{3}$

Plot the graph of an input expression

In [43]:
```
from sympy import Symbol, sympify, solve
from sympy.plotting import plot

def plot_expression(expr):

  y = Symbol('y')
  solutions = solve(expr,y)
  expr_y = solutions[0]
```

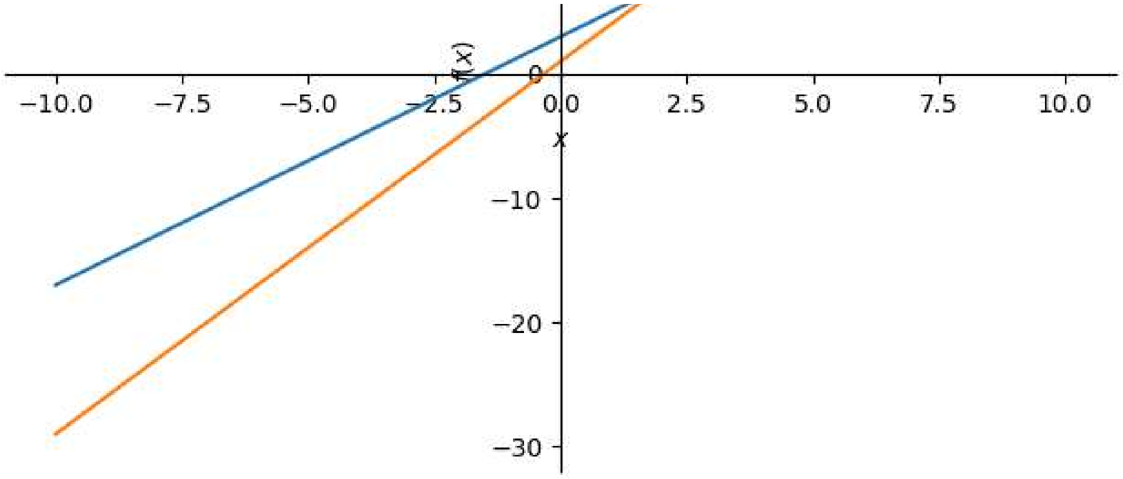
```
  plot(expr_y)


if __name__ == '__main__':
  expr = input('enter your expression in terms of x and y:')
  try :
    expr = sympify(expr)
  except SympifyError:
    print('invalid input')           #2*x + 3*y -6
  else:
    plot_expression(expr)
```

```
enter your expression in terms of x and y: 2*x + 3*y -6
```



Plotting Multiple Functions

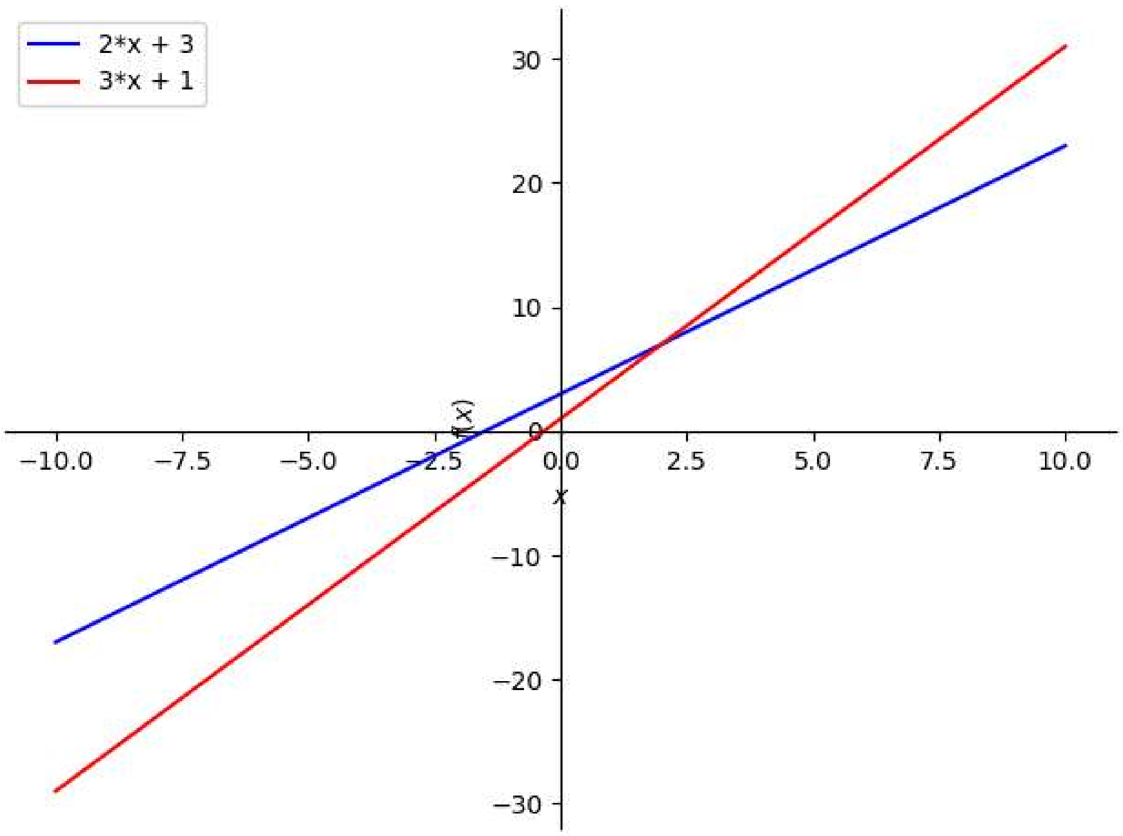
In [44]:
```
from sympy.plotting import plot
from sympy import Symbol
x =Symbol('x')
plot(2*x+3, 3*x+1)
```



Out[44]: <sympy.plotting.backends.matplotlibbackend.matplotlib.MatplotlibBackend at 0x7bd9e43f8cd0>

In [45]:

```
from sympy.plotting import plot
from sympy import Symbol
x = Symbol('x')
p = plot(2*x+3, 3*x+1, legend = True, show = False)
p[0].line_color = 'b'
p[1].line_color ='r'
p.show()
```



Programming challenges

1. Factor Finder

2.Graphical Equation Solver

In [46]:
```
expr1 = input('enter your first expression in terms of x and y:')
expr2 = input('enter your second expression in terms of x and y:')
```

```
enter your first expression in terms of x and y: #2*x + 3*y -6
enter your second expression in terms of x and y: #2*x + 3*y -6
```

summing a series

In [47]:
```
"""for i in range(2, n+1):
  series = series + (x**i)/i """
```

Out[47]: `'for i in range(2, n+1):\n  series = series + (x**i)/i '`

In [48]:
```
from sympy import Symbol, summation , pprint
x = Symbol('x')
n = Symbol('n')
s = summation(x**n/n, (n,1,5))
pprint(s)
```

```
     2    3    4    5
    x    x    x    x
x + ── + ── + ── + ──
    2    3    4    5
```

In [49]:
```
s.subs({x:1.2})
```

Out[49]: 3.512064

In [50]:
```
def sum_of_series(nth_term, num_terms):
  """
  Calculates the sum of a series given the nth term and the number of terms.

  Args:
    nth_term: A string representing the nth term of the series.
    num_terms: An integer representing the number of terms in the series.

  Returns:
    A string representing the sum of the series.
  """

  # Replace 'n' with the term number in the nth term expression
  sum_expression = ""
  for i in range(1, num_terms + 1):
    term_expression = nth_term.replace('n', str(i))
    if i > 1:
      sum_expression += " + "
    sum_expression += term_expression

  return sum_expression

# Example usage
nth_term = "a+(n-1)*d"
num_terms = 3
```

```
sum_of_series_str = sum_of_series(nth_term, num_terms)
print("Sum of the series:", sum_of_series_str)
```

```
Sum of the series: a+(1-1)*d + a+(2-1)*d + a+(3-1)*d
```

solving single variable inequalities

In [51]:

```
from sympy import poly, Symbol, solve_poly_inequality
x = Symbol('x')
ineq_obj = -x**2 + 4 < 0
lhs = ineq_obj.lhs
p = poly(lhs,x)
rel = ineq_obj.rel_op
solve_poly_inequality(p,rel)
```

Out[51]: $[(-\infty, -2), \ (2, \infty)]$

In [52]:

```
from sympy import Symbol, poly, solve_rational_inequalities
x = Symbol('x')
ineq_obj = ((x-1)/(x+2)) > 0
lhs = ineq_obj.lhs
numer, denom = lhs.as_numer_denom()
p1 = poly(numer)
p2 = poly(denom)
rel = ineq_obj.rel_op
solve_rational_inequalities([[((p1,p2),rel)]])
```

Out[52]: $(-\infty, -2) \cup (1, \infty)$

In [53]:

```
from sympy import Symbol, solve, solve_univariate_inequality, sin
x = Symbol('x')
ineq_obj = sin(x)-0.6 > 0
solve_univariate_inequality(ineq_obj, x, relational=False)
```

Out[53]: $(0.643501108793284, \pi - 0.643501108793284)$

Hints: Handy Functions

Now remember—your challenge is (1) to create a function, isolve(), that will take any inequality and (2) to choose one of the appropriate functions discussed in this section to solve it and return the solution. The following hints may be useful to implement this function.

The is_polynomial() method can be used to check whether an expression is a polynomial or not:

In [54]:

```
x = Symbol('x')
expr = x**2 - 4
expr.is_polynomial(x)
```

Preview Code Blame Raw

Out[55]: False

In [56]:
```
expr = (2+x)/(3+x)
expr.is_rational_function()
```

Out[56]: True

In [57]:
```
expr = 2+x
expr.is_rational_function()
```

Out[57]: True

In [58]:
```
expr= 2+sin(x)
expr.is_rational_function()
```

Out[58]: False

In [59]:
```
from sympy import sympify
sympify('x+3>0')
```

Out[59]: $3 + x > 0$

chapter ends.