python-intro

March 19, 2021

0.1 python version

- 2.7
- 3.9

```
python --version
```

1 Hello world

[77]:
```
# print("HELLO WORLD")

print("hello world")
```

```
hello world
```

[78]:
```
print("hello world")

print("-------------------------")

print("Hello AI")
```

```
hello world
-------------------------
Hello AI
```

[79]:
```
#Multiple Statements on a Single Line
print("hello world"); print("-------------------------"); print("Hello AI")
```

```
hello world
-------------------------
Hello AI
```

2 variables

2.1 name

- keywords `help('keywords')`
- Camel-Case, Pascal-case, Hungarian

[80]:
```
help('keywords')
```

```
Here is a list of the Python keywords.  Enter any keyword to get more help.

False               break               for                 not
None                class               from                or
True                continue            global              pass
__peg_parser__      def                 if                  raise
and                 del                 import              return
as                  elif                in                  try
assert              else                is                  while
async               except              lambda              with
await               finally             nonlocal            yield
```

[81]:
```
#Camel-Case
studentAge = 20
student_age = 20

#Pascal
StudentAge = 20

#Hungarian
i_studentAge = 20

print(student_age)
print(type(student_age))
```

```
20
<class 'int'>
```

[ ]:
```

```

2.2 type

- Numbers: int, float, uint, double…
- Boolean
- String
- List
- Tuple
- Dictionary

2.2.1 numbers and Arithmetic Operators: + - * ** / // %

[82]:
```
# numbers
# Arithmetic Operators: + - * ** / // %
student_age = 20

print(type(student_age))

house_price = 8.3

print(type(house_price))
```

```
<class 'int'>
<class 'float'>
```

[83]:
```
a = student_age + house_price

print(type(a))
```

```
<class 'float'>
```

[84]:
```
2**3
```

[84]: 8

[85]:
```
7 % 4
```

[85]: 3

2.2.2 bool and Comparison Operators

- == If the values of two operands are equal, then the condition becomes true. (a == b) is not true.
- != If values of two operands are not equal, then condition becomes true. (a != b) is true.
- <> If values of two operands are not equal, then condition becomes true. (a <> b) is true. This is similar to != operator.
- < If the value of left operand is less than the value of right operand, then condition becomes true. (a < b) is true.
- >= If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. (a >= b) is not true.
- <= If the value of left operand is less than or equal to the value of right operand, then condition becomes true. (a <= b) is

[86]:
```
a = False
b = True

c = 5
d = 9
```

```
c = c < d
print(c)
```

```
True
```

3 condition control

[87]:
```
lowest_price = 2
current_price = 3
condition = current_price < lowest_price
if (condition):
    print("True") # Indentation
else:
    print("False")
```

```
False
```

3.1 Indentation

- Indentation is to determine the grouping of statements.
- one `tab` or `four spaces`

[88]:
```
condition = False
if (condition):
    print("True")
    print("code block for True")
    print("indent is used to identify code blocks")
    print("common indent: four space")
else:
    print("False")
    print("code block for False")
```

```
False
code block for False
```

[89]:
```
lowest_price = 2
current_price = 1
condition = current_price < lowest_price
if (condition):
    print("True")
    lowest_price = current_price
else:
    print("False")
print(lowest_price)
```

```
True
1
```

4 list

- The list is written as a list of comma-separated values (items) between square brackets.
- Important thing about a list is that items in a list need not be of the same type.
- the index of list items starts from 0.

[90]:
```
a = [5, 3, 1.3, 0, -5, "AI"]
print(a)
```

```
[5, 3, 1.3, 0, -5, 'AI']
```

[91]:
```
# Accessing Values in Lists by index
print(a[0], a[2], a[-2])
```

```
5 1.3 -5
```

[92]:
```
# length of list
# last item
# len, -1

print("length of list a is ", len(a))
print("the last item in list a is ", a[len(a) - 1], a[-1])
```

```
length of list a is  6
the last item in list a is  AI AI
```

4.1 slicing a[start_idx : end_idx]

[93]:
```
# slicing
# [start, end )
print(a[2 : 4])
print(a[-4 : -2])
```

```
[1.3, 0]
[1.3, 0]
```

[94]:
```
print(a[: 4])
print(a[2: ])
```

```
[5, 3, 1.3, 0]
[1.3, 0, -5, 'AI']
```

[95]:
```
print(a[-4: ])
print(a[: -2])
```

```
[1.3, 0, -5, 'AI']
[5, 3, 1.3, 0]
```

[96]:
```
print(a[4 : 2])
print(a[-2 : -4])
```

```
[]
[]
```

4.2 slicing with skip a[start_idx : end_idx : skip]

[97]:
```
print(a[ 1: 5: 2])
print(a[ 1: 5: 3])
```

```
[3, 0]
[3, -5]
```

[98]:
```
print(a[ 1: 5: 4])
```

```
[3]
```

[99]:
```
print(a[ 1: 6: 4])
```

```
[3, 'AI']
```

4.3 change list

[100]:
```
a.append(10)
print(a)
```

```
[5, 3, 1.3, 0, -5, 'AI', 10]
```

[101]:
```
a.insert(2, 10)
print(a)
```

```
[5, 3, 10, 1.3, 0, -5, 'AI', 10]
```

[102]:
```
a.remove(-5)
print(a)
```

```
[5, 3, 10, 1.3, 0, 'AI', 10]
```

[103]:
```
a.remove(10)
print(a)
```

```
[5, 3, 1.3, 0, 'AI', 10]
```

4.4 some useful function of list

- max()
- min()
- len()

[104]:
```
price_list = [9,7,4,6,7,8,6,8,4,7,5]
max(price_list), min(price_list)
```

[104]:
```
(9, 4)
```

5 for loop

[105]:
```
price_list = [9,7,4,6,7,8,6,8,4,7,5]
lowest_price = 1000
for current_price in price_list:
    condition = current_price < lowest_price
    if (condition):
        print("True")
        lowest_price = current_price
    else:
        print("False")
print(lowest_price)
```

```
True
True
True
False
False
False
False
False
False
False
False
4
```

[106]:
```
# for loop
price_list = [9,7,4,6,7,8,6,8,4,7,5]
lowest_price = 1000
for idx in range(len(price_list)):
    current_price = price_list[idx]
    condition = current_price < lowest_price
    if (condition):
        print("True")
        lowest_price = current_price
    else:
        print("False")
print(lowest_price)
```

```
True
True
True
```

```
False
False
False
False
False
False
False
False
4
```

5.1 tuple

- The tuple is written as a list of comma-separated values (items) between parentheses.
- The tuple cannot be modified.

[107]:
```
a = (1, 2, 3, 4)
print(type(a))
print(a, len(a))
print(a[0], a[-1])
print(max(a), min(a))
```

```
<class 'tuple'>
(1, 2, 3, 4) 4
1 4
4 1
```

[108]:
```
list_a = list(a)
print(list_a)
list_a.append(3)
print(list_a)
print(a)

b = tuple(list_a)
print(b)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 3]
(1, 2, 3, 4)
(1, 2, 3, 4, 3)
```

5.2 Functions

Python functions are defined using the `def` keyword.

[109]:
```
def greet(s):
    print("hello, ", s)
greet("Cat")
```

```
hello,  Cat
```

[110]:
```
def greet(s):
    print("hello, ", s)
    return s*2

c = greet(3)
print(c)
```

```
hello,  3
6
```

5.3 example: univariate linear model

$a = wx + b$

for example

$a = 2x + 3$

[111]:
```
#one argument
def linear(x):
    a = 2*x+3
    return a

for x in [-1, 0, 1]:
    print(linear(x))
```

```
1
3
5
```

[112]:
```
#three arguments
def linear(x, w, b):
    a = w*x+b
    return a

for x in [-1, 0, 1]:
    print(linear(x, 2, 3))
```

```
1
3
5
```

[113]:
```
#argument with default values
def linear(x, w=2, b=3):
    a = w*x+b
    return a

for x in [-1, 0, 1]:
    print(linear(x, 2, 3))
```

```
print('-'*20)

for x in [-1, 0, 1]:
    print(linear(x))
```

```
1
3
5
--------------------
1
3
5
```

[114]:
```
#argument with default values
def linear(x, w=2, b):
    a = w*x+b
    return a
```

```
  File "<ipython-input-114-df70d8d50ceb>", line 2
    def linear(x, w=2, b):
                       ^
SyntaxError: non-default argument follows default argument
```

6 module

[115]:
```
import os
print(os)

print('-'*20)

print(os.listdir())
```

```
<module 'os' from 'C:\\Programs\\Python\\lib\\os.py'>
--------------------
['.ipynb_checkpoints', '.vscode', 'homework.ipynb', 'images', 'lecture.ipynb',
'lecture2.ipynb', 'Lecture2.pdf', 'Lecture2.pptx', 'python-intro.html', 'python-
intro.ipynb', 'python-intro.pdf', 'test.py']
```

[116]:
```
from os import listdir
listdir()
```

[116]:
```
['.ipynb_checkpoints',
 '.vscode',
 'homework.ipynb',
 'images',
```

```
 'lecture.ipynb',
 'lecture2.ipynb',
 'Lecture2.pdf',
 'Lecture2.pptx',
 'python-intro.html',
 'python-intro.ipynb',
 'python-intro.pdf',
 'test.py']
```

[117]:
```
from os import listdir as listdir_os
listdir_os()
```

[117]:
```
['.ipynb_checkpoints',
 '.vscode',
 'homework.ipynb',
 'images',
 'lecture.ipynb',
 'lecture2.ipynb',
 'Lecture2.pdf',
 'Lecture2.pptx',
 'python-intro.html',
 'python-intro.ipynb',
 'python-intro.pdf',
 'test.py']
```

6.1 Matplotlib

Matplotlib is a plotting library.

`import matplotlib.pyplot as plt`

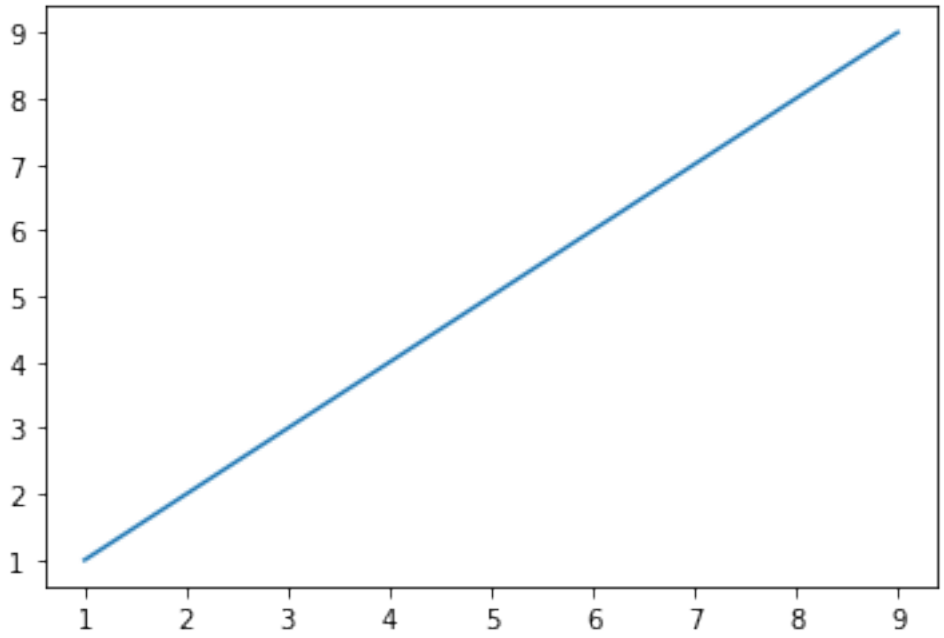
allows you to plot 2D data.

install with `pip install matplotlib`

[118]:
```
import matplotlib.pyplot as plt
```

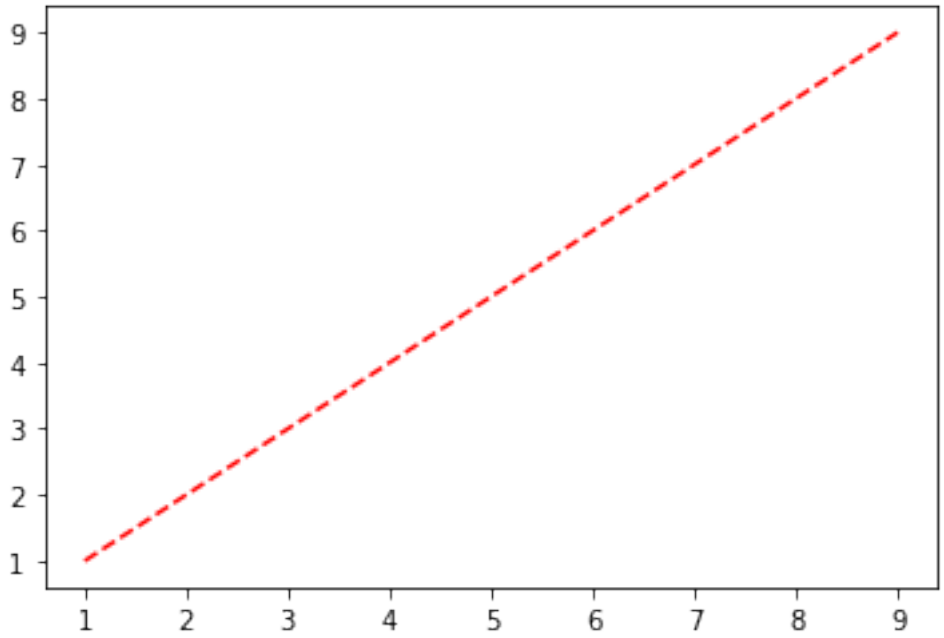
[119]:
```
x = [1,2,5,8,9]
y = [1,2,5,8,9]

plt.plot(x, y, '-') #shit + tab to show the doc
```

[119]:
```
[<matplotlib.lines.Line2D at 0x24140773f10>]
```



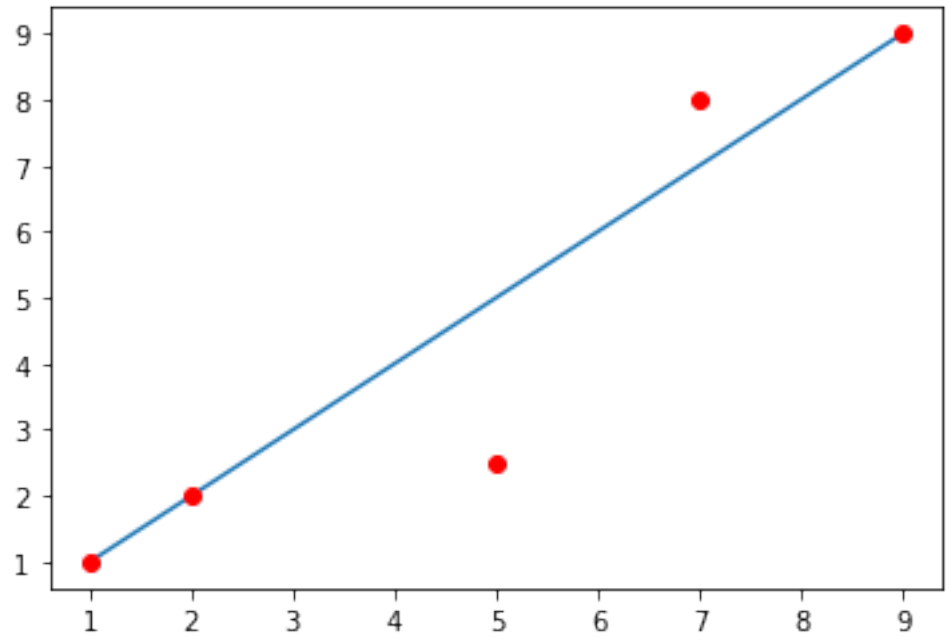
[120]:
```
plt.plot(x, y, 'r--') #shit + tab to show the doc
```

[120]:
```
[<matplotlib.lines.Line2D at 0x241407d9970>]
```



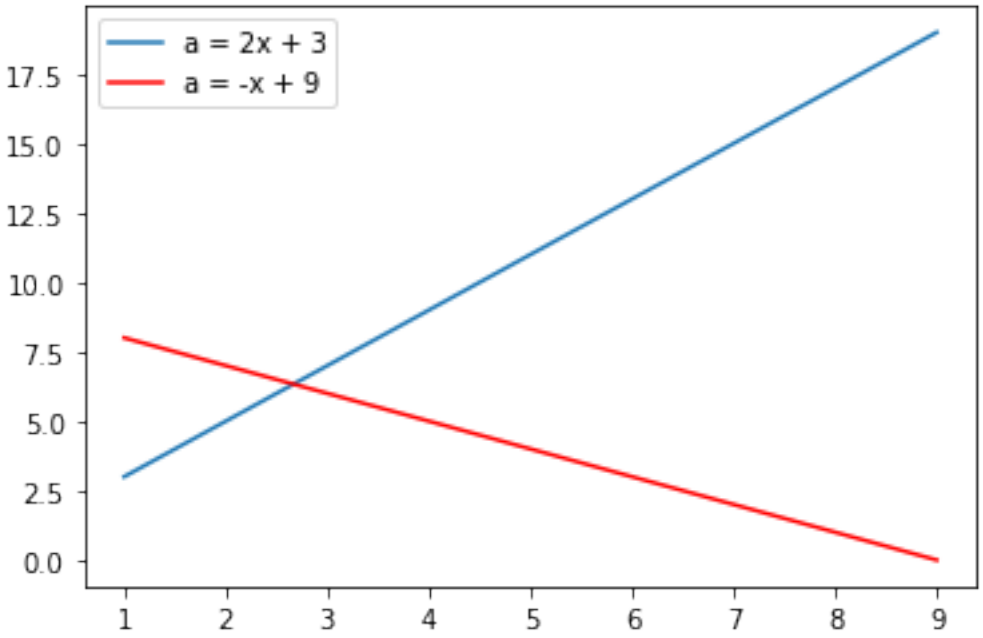
[121]:
```
x = [1,2,5,8,9]
y = [1,2,5,8,9]

x_noise = [1,2,5,7,9]
y_noise = [1,2,2.5,8,9]

plt.plot(x, y, '-')
plt.plot(x_noise, y_noise, 'ro')
```

[121]: [<matplotlib.lines.Line2D at 0x2414084c730>]



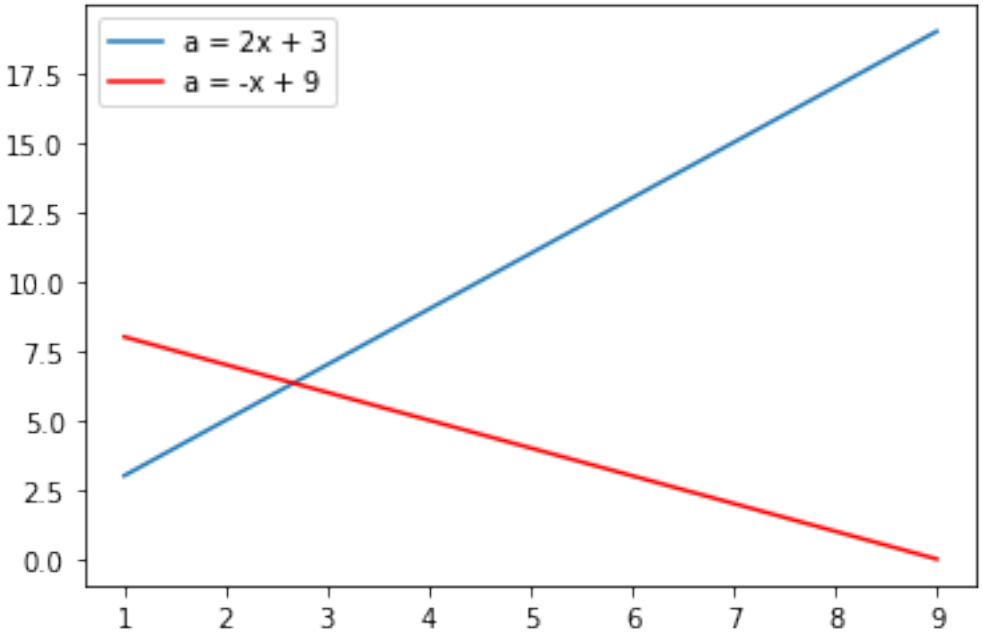
[122]:
```
#a = 2x + 3

x = [1,2,5,8,9]

a1 = [linear(x_i, 2, 1) for x_i in x]

#a = -x + 9
a2 = [linear(x_i, -1, 9) for x_i in x]

plt.plot(x, a1, '-')
plt.plot(x, a2, 'r-')
plt.legend(['a = 2x + 3', 'a = -x + 9'])
```

[122]: <matplotlib.legend.Legend at 0x241408a7460>

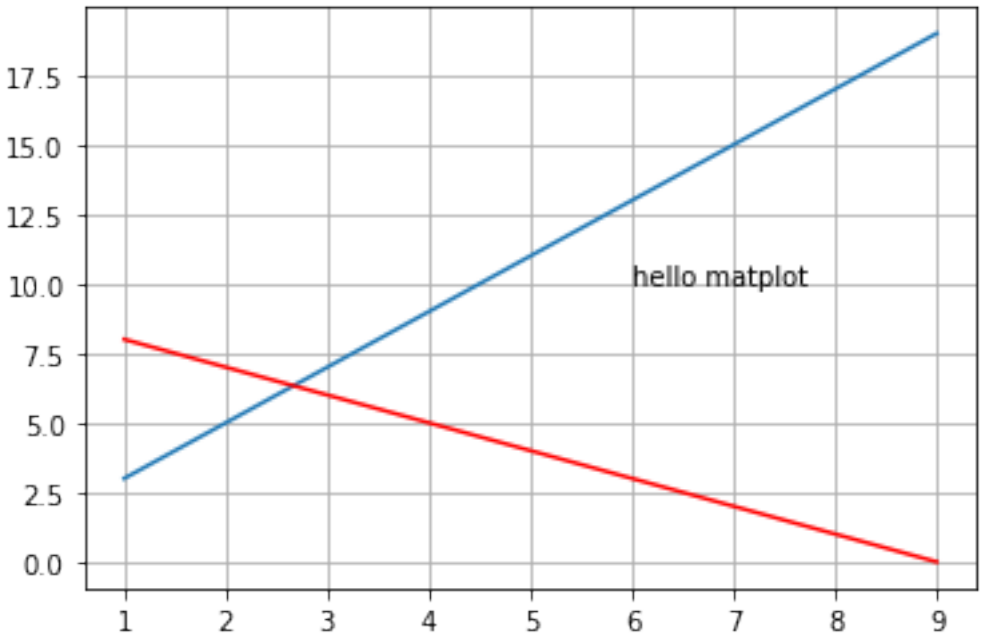


[123]:
```
plt.plot(x, a1, '-', x, a2, 'r-') #
plt.legend(['a = 2x + 3', 'a = -x + 9'])
```

[123]: <matplotlib.legend.Legend at 0x24140882f70>

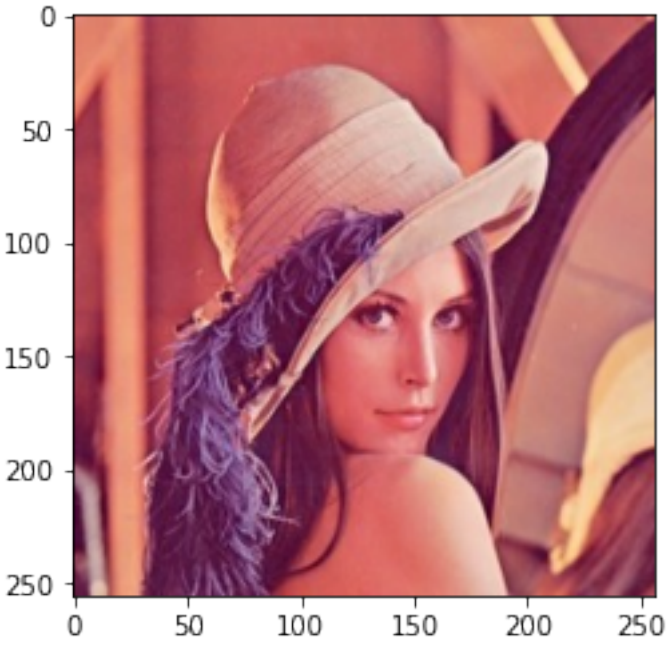


[124]:
```
plt.plot(x, a1, '-', x, a2, 'r-') #
plt.text(6,10,"hello matplot")
plt.grid()
```



[125]:
```
from PIL import Image

img = Image.open("images/lena.jpg")
plt.imshow(img)
```

[125]: <matplotlib.image.AxesImage at 0x241419d6250>

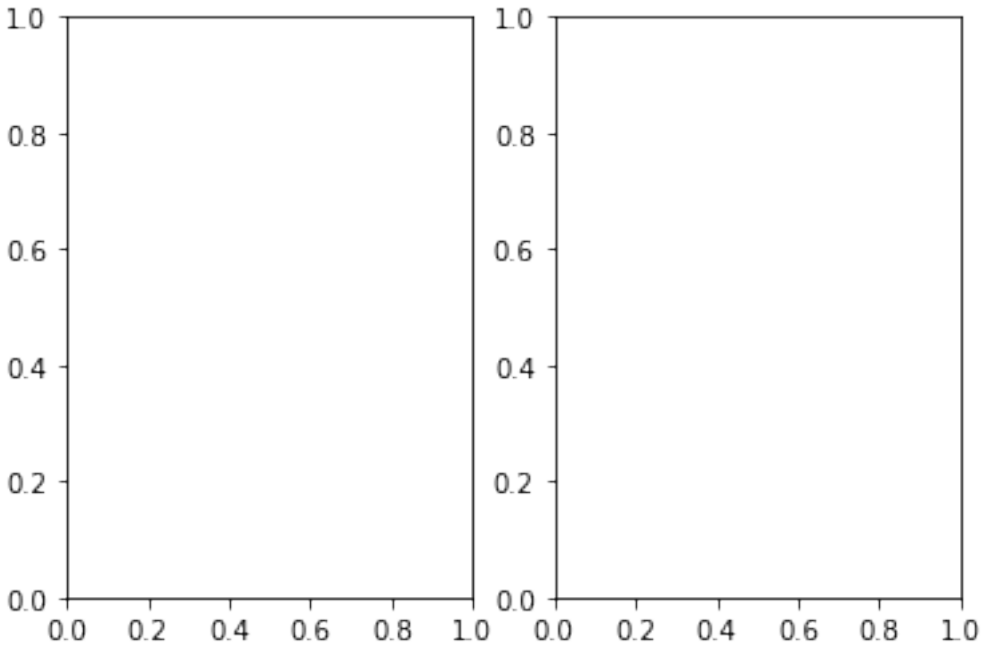


6.1.1 Subplots

You can plot different things in the same figure using the subplot function.

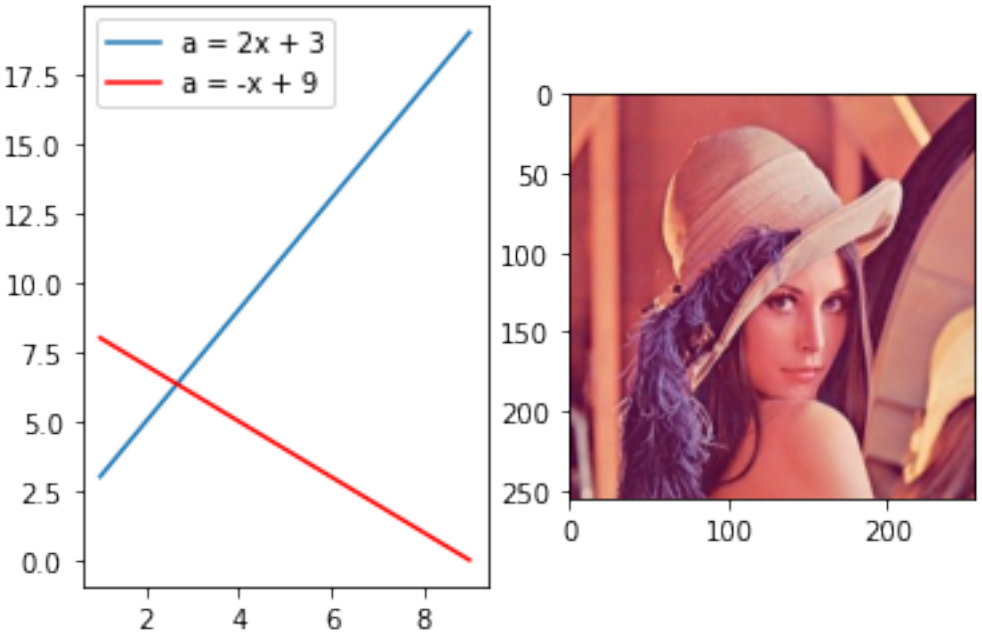
[126]:
```
plt.subplot(1,2,1)
plt.subplot(1,2,2)
```

[126]:
```
<AxesSubplot:>
```



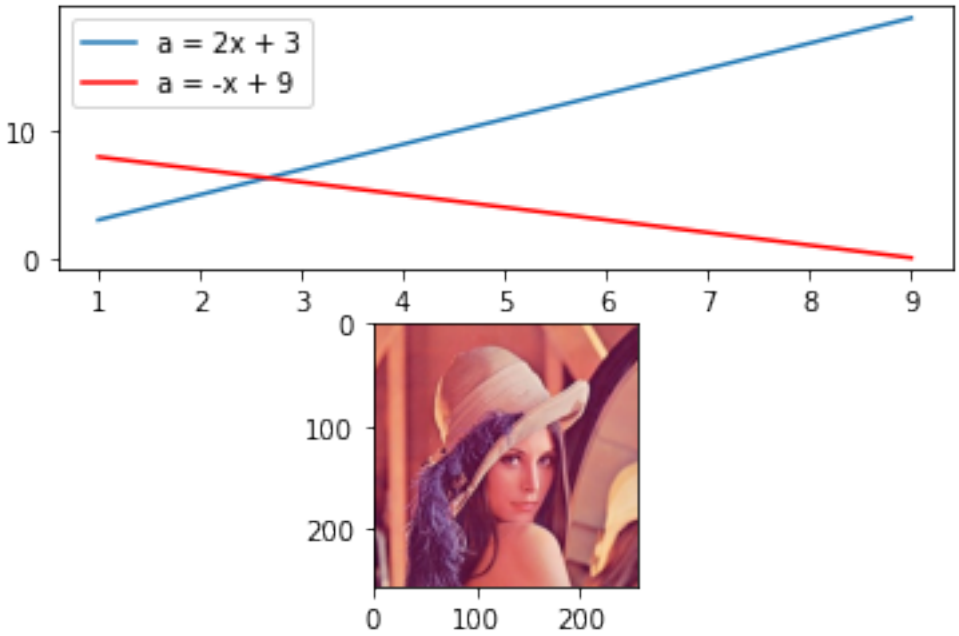
[127]:
```
plt.subplot(1,2,1)
plt.plot(x, a1, '-', x, a2, 'r-') #
plt.legend(['a = 2x + 3', 'a = -x + 9'])


plt.subplot(1,2,2)
img = Image.open("images/lena.jpg")
plt.imshow(img)
```

[127]: <matplotlib.image.AxesImage at 0x24141ac59d0>



[128]:
```
plt.subplot(2,1,1)
plt.plot(x, a1, '-', x, a2, 'r-') #
plt.legend(['a = 2x + 3', 'a = -x + 9'])


plt.subplot(2,1,2)
img = Image.open("images/lena.jpg")
plt.imshow(img)
```

[128]: <matplotlib.image.AxesImage at 0x24141b75280>


a = 2x + 3
a = -x + 9
10
0
1
2
3
4
5
6
7
8
9

0
100
200
0
100
200