Ploting with the matplotlib library

In this lecture we will create graphs with the matplotlib library [wiki]

Before we begin, we should mention that "plotting" does not traditionally belong to the course of programming, but to that of data analysis. However, programming in biology and in science in general usually contains data analysis.

Plotting is art!

A good plot has nothing to do with being a good programmer or having qualitative data (although both help). Good plotting has to do with your aesthetics, your perception of colors and your sense of proportion.

That is why before (or even after ...) this lecture you should take a look at the following:

- Ten Simple Rules for Better Figures. Quote: "All the figures for this article were produced using matplotlib, and figure scripts are available from https://github.com/rougier/ten-rules ."
- Visual Analytics in Omics: why, what, how? by Jan Aerts
- About image manipulation: Creating clear and informative image-based figures for scientific publications
- Matplotlib for papers (Maybe this is a little old)
- Get inspired by the very good plots of tableau

Also you should take a look of how **NOT** to make plots:

- Misleading graph
- Google: sky chart

The usual way to import matplotlib is:

In [2]:
```
import matplotlib.pyplot as plt
```

Then for each new plot we must create the following two objects fig and ax

In [4]:
```
fig, ax = plt.subplots()
```

In general: matplotlib has three basic items for handling plots:

- ax: Axis handling, most of the time we will deal with this object.
- fig: Handling the plot as an image.
- plt: This is the pyplot object we have imported. Contains the basic methods of ax and fig object. It exists as an auxiliary object, simplification so that one does not have to "play" with two objects. e.g. one can write: `ax.plot()` or `plt.plot()` similarly one can write `fig.show()` or `plt.show()` . However there are methods that are part of the ax object and not of the plt object. And since python is a

language "There should be one-- and preferably only one --obvious way to do it." the existence of these choices I consider to be outside the spirit of language (they are mainly historical reasons why this is done). That's why we will not deal that much with plt!

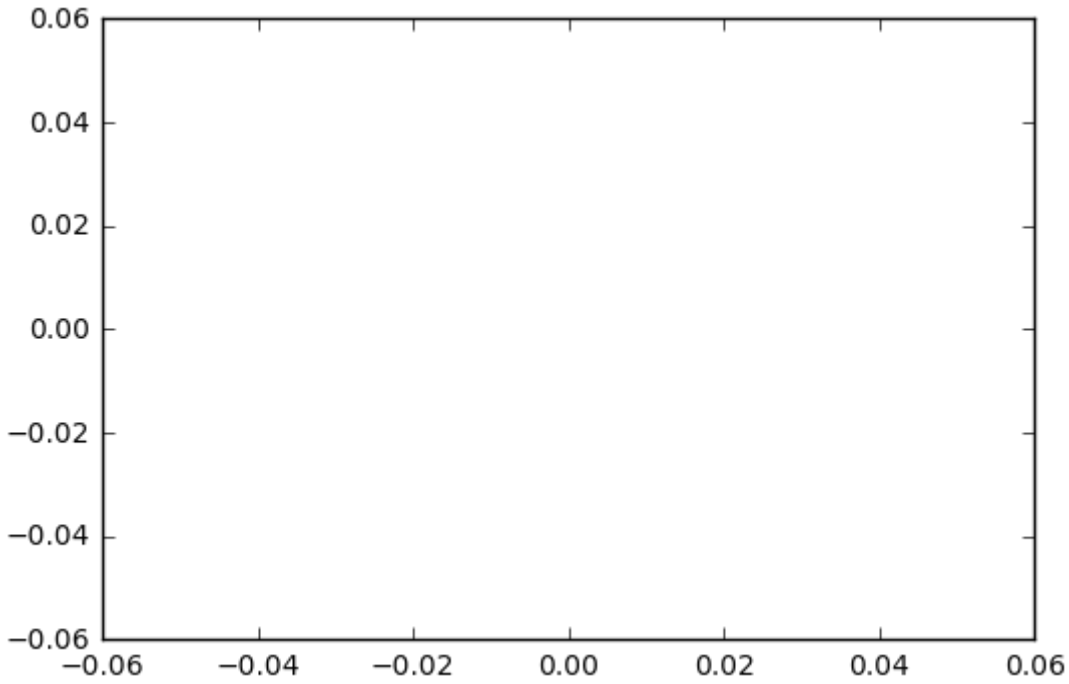
Let's make an empty plot!

In [5]:
```
ax.plot()
```

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

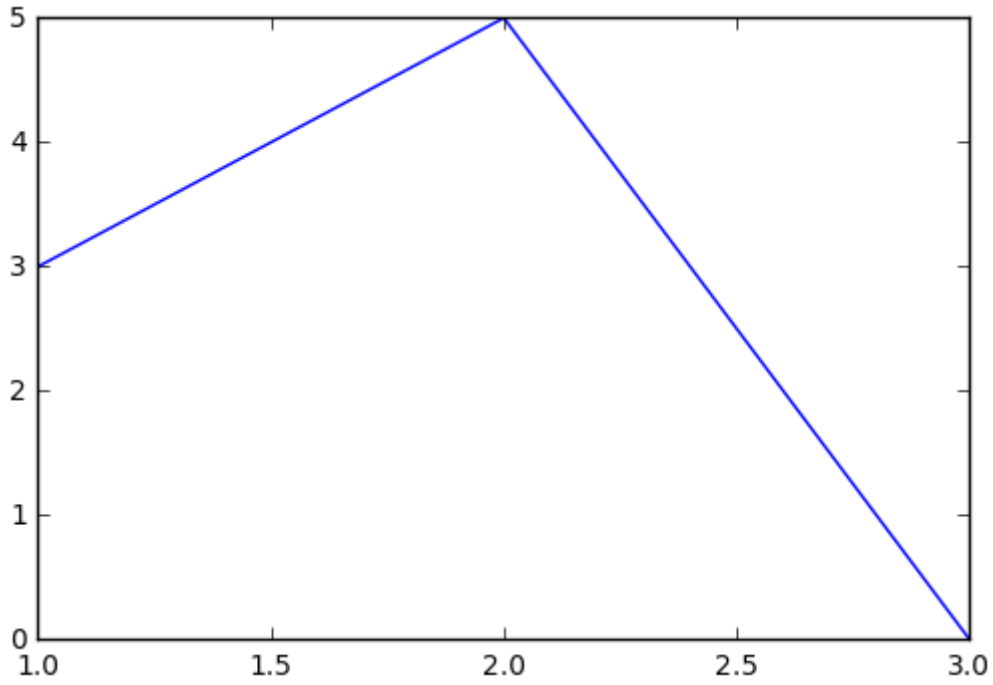
To display a plot we use the `show()` method:

In [7]:
```
plt.show()
```



`ax.plot` accepts a wide variety of arguments. The first two arguments are lists. The first contains the X-axis coordinates of the elements we want to plot, and the second the Y-axis coordinates, e.g. To display a zigzag line passing through the points: (1,3), (2,5), (3,0):

In [11]:
```
fig, ax = plt.subplots()
ax.plot([1,2,3], [3,5,0])
plt.show()
```



`plot` also accepts a third argument which is the "style" of the line. It consists of two parts: color and style. The color can be (http://matplotlib.org/api/colors_api.html):

- b: blue
- g: green
- r: red
- c: cyan
- m: magenta
- y: yellow
- k: black
- w: white

In case we want to plot a line then the style can be either one of (http://matplotlib.org/api/lines_api.html#matplotlib.lines.Line2D.set_linestyle):

- `-` or `solid` solid line
- `--` or `dashed` dashed line
- `-.` or `dashdot` dash-dotted line
- `:` or `dotted` dotted line
- `None` draw nothing
- ` ` draw nothing
- `''` draw nothing

In case we want to plot only the points (and not lines) then the options are (http://matplotlib.org/api/markers_api.html):

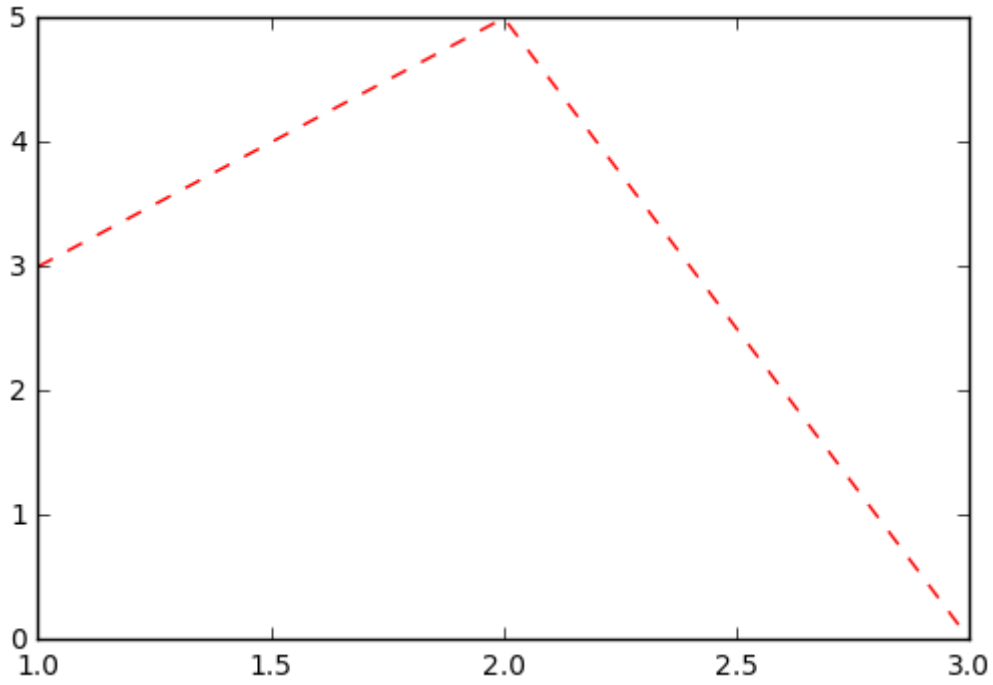
- ".” point
- ",” Pixel
- “O” circle
- “V” triangle_down
- “^” Triangle_up
- “<” Triangle_left

- “>” Triangle_right
- “1” tri_down
- “2” tri_up
- “3” tri_left
- “4” tri_right
- "8" octagon
- “S” square
- "P" pentagon
- “*” Star
- “H” hexagon1
- “H” hexagon2
- “+” Plus
- “X” x
- “D” diamond
- “D” thin_diamond
- “|” vline
- “_” Hline
- TICKLEFT tickleft
- TICKRIGHT tickright
- TICKUP tickup
- TICKDOWN tickdown
- CARETLEFT caretleft
- CARETRIGHT caretright
- CARETUP caretup
- CARETDOWN caretdown
- "None" nothing
- None nothing
- ”“ Nothing
- "" Nothing

  There are more options and possibilities for "markers" http://matplotlib.org/api/markers_api.html

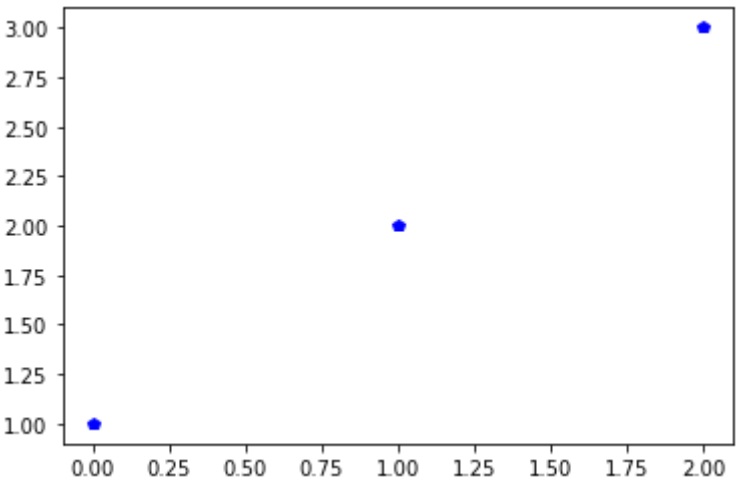
For example a red dotted line:

In [14]:
```
fig, ax = plt.subplots()
ax.plot([1,2,3], [3,5,0], 'r--')
plt.show()
```



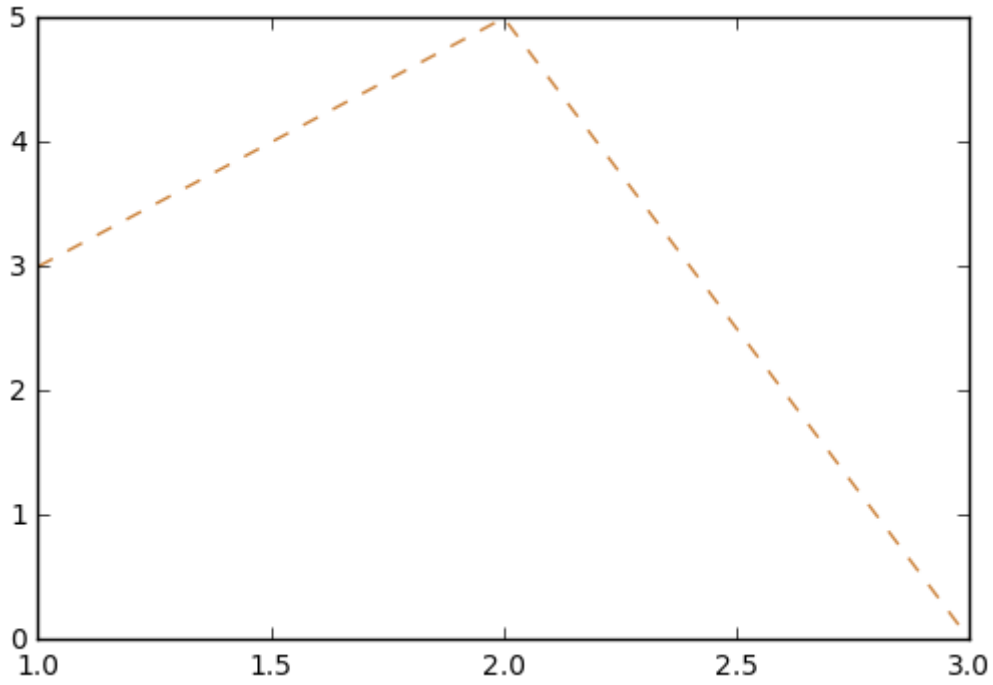
Markers as blue pentagons:

In [3]:
```
fig, ax = plt.subplots()
ax.plot([1,2,3], 'bp') # Points are: (1,1), (2,2), (3,3). bp: b=blue, p=pe
plt.show()
```



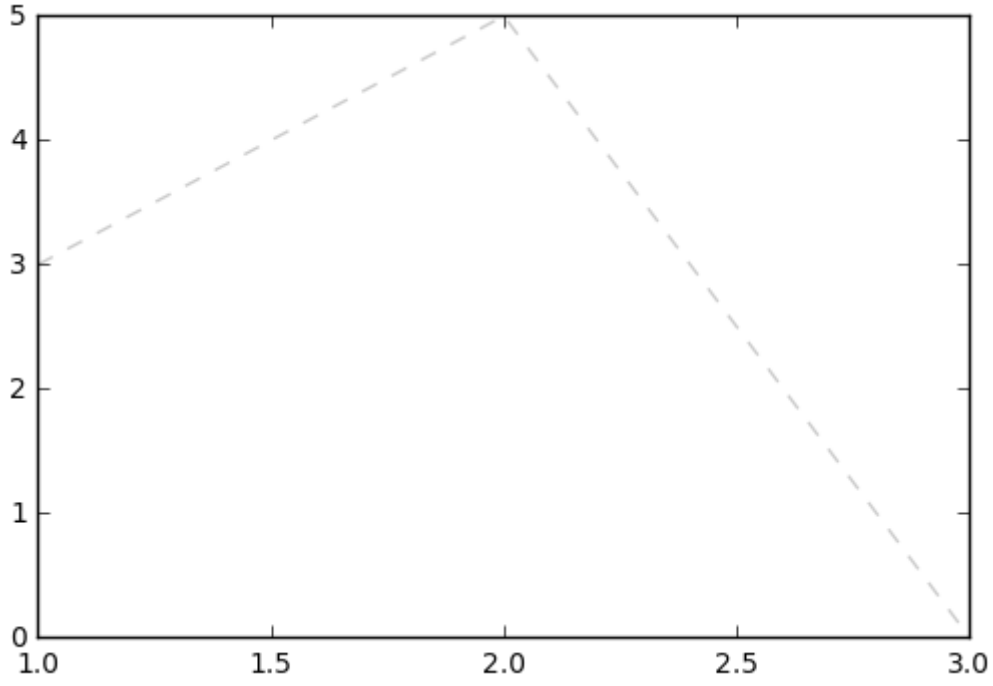
Of course there is a better way to define a color, with the parameter `c` :

In [19]:
```
fig, ax = plt.subplots()
ax.plot([1,2,3], [3,5,0], '--', c="peru")
plt.show()
```



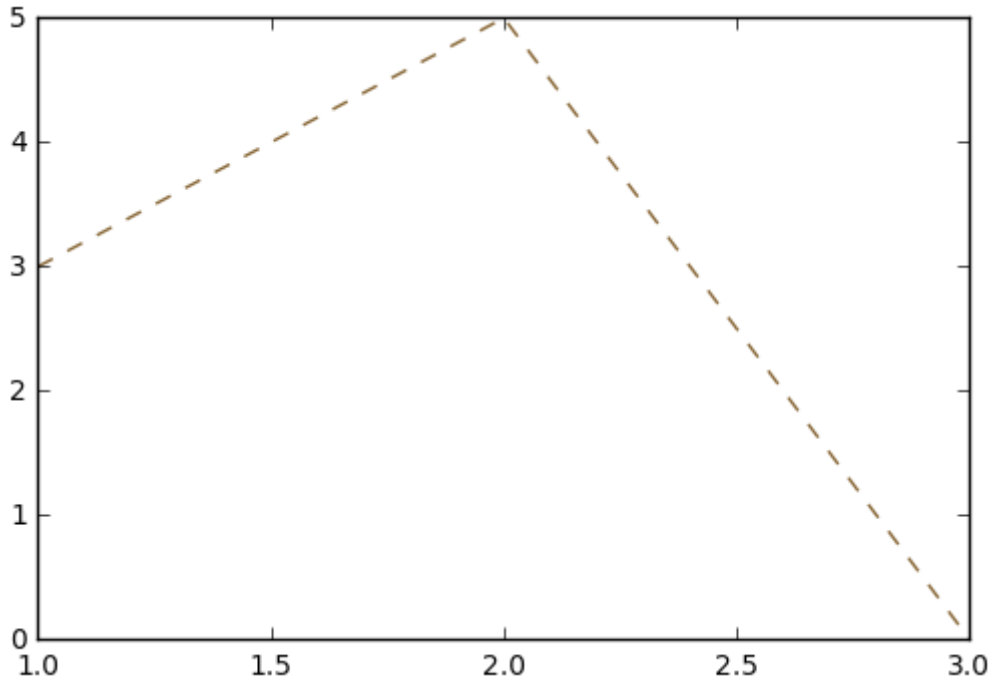
Yes, there is a color called "peru". A full list of names can be found here: (http://matplotlib.org/examples/color/named_colors.html). Alternatively you can use a value from 0.0 to 1.0 to print to a "grayscale" where 0.0 is black and 1.0 is white:

In [20]:
```
fig, ax = plt.subplots()
ax.plot([1,2,3], [3,5,0], '--', c="0.8")
plt.show()
```



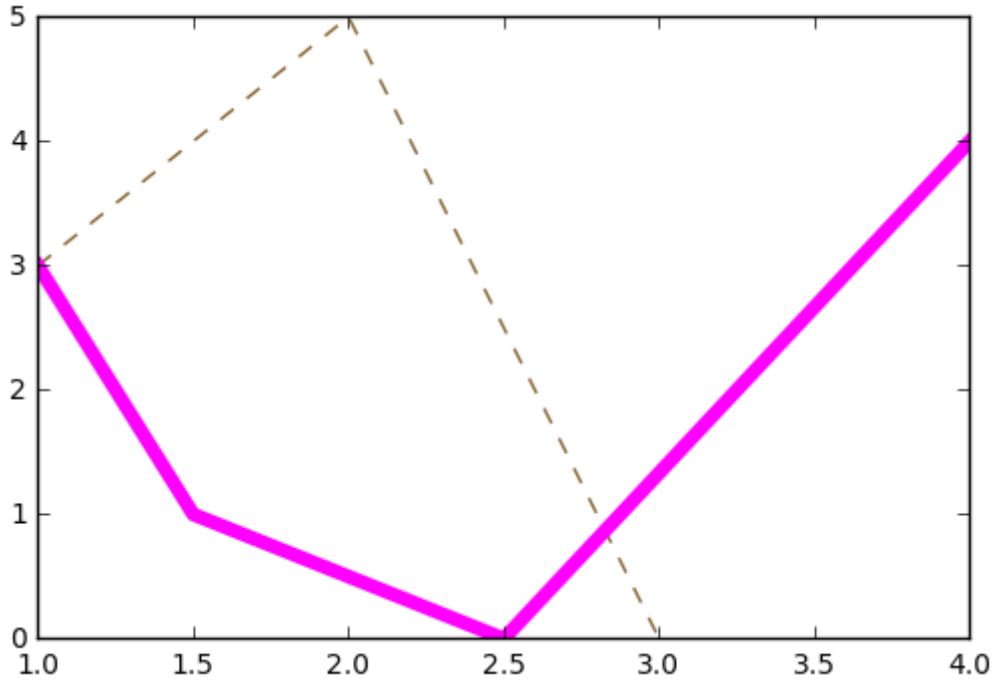
Of course you can use any RGB color . There are many sites where you can choose a color eg: http://htmlcolorcodes.com/

In [21]:
```
fig, ax = plt.subplots()
ax.plot([1,2,3], [3,5,0], '--', c="#876635")
plt.show()
```



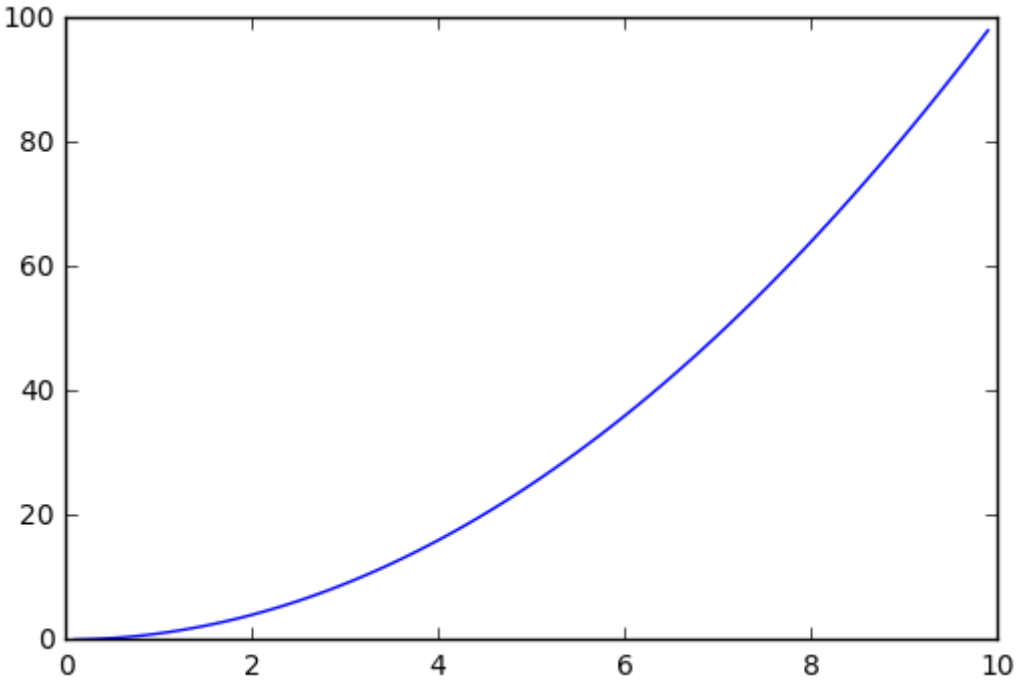
Plot can be used many times:

In [24]:
```
fig, ax = plt.subplots()
ax.plot([1,2,3], [3,5,0], '--', c="#876635")
ax.plot([1, 1.5, 2.5, 4], [3,1, 0, 4], '-', c="magenta", linewidth=5) # Pa
plt.show()
```



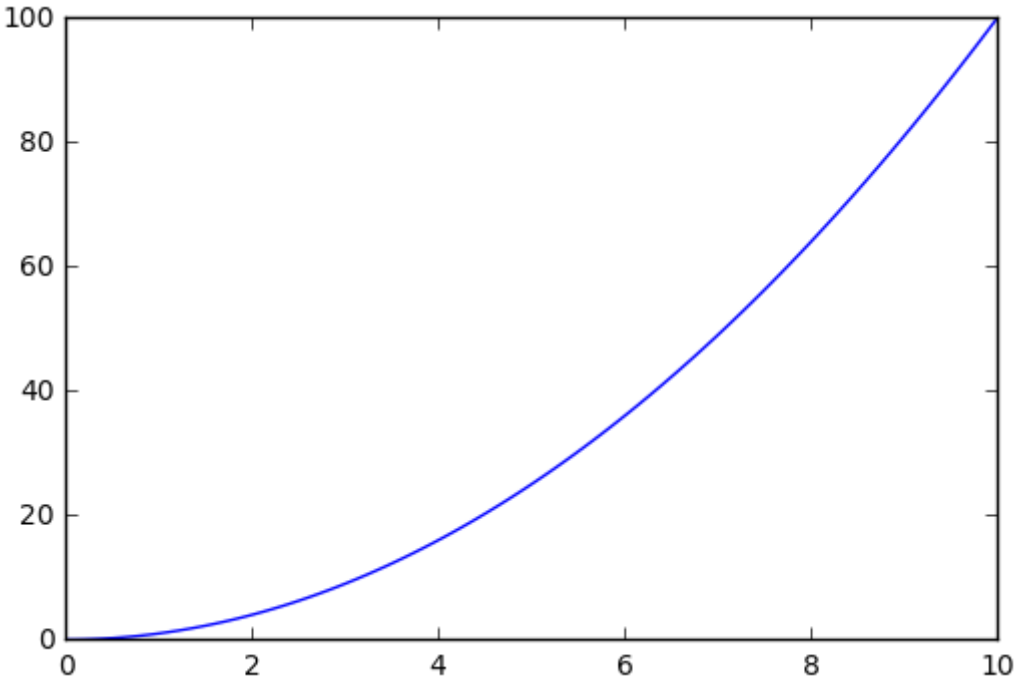
We can also plot a function `f` by computing `x`, and `y=f(x)`:

In [25]:
```
def f(x):
    return x**2 # X square

X = [x/10.0 for x in range(1,100)]
Y = [f(x) for x in X]

fig, ax = plt.subplots()
ax.plot(X,Y)
plt.show()
```



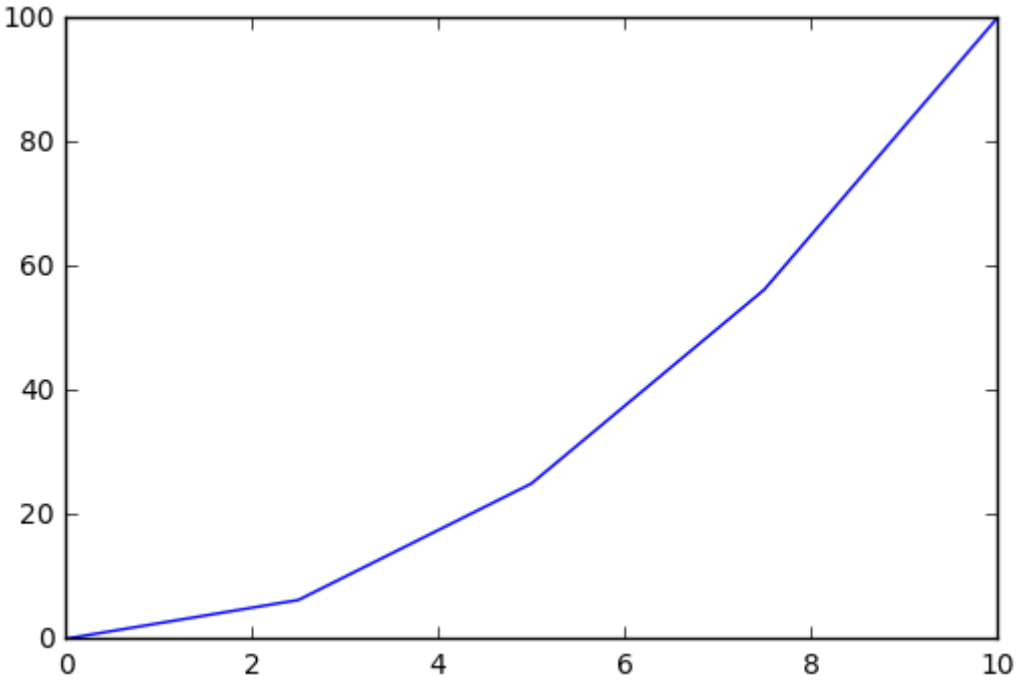
A better way to plot functions is to use numpy 's linspace. The linspace (a, b, c) creates an arithmetic progression from a to b, so that there are a total of c elements.

In [27]:
```
import numpy as np

X = np.linspace(0, 10, 100) # 0 = min X, 10 = max X, 100 = resolution...
Y = [f(x) for x in X]

fig, ax = plt.subplots()
ax.plot(X,Y)

plt.show()
```



The value 100 in linspace can also be seen as the "graph resolution":

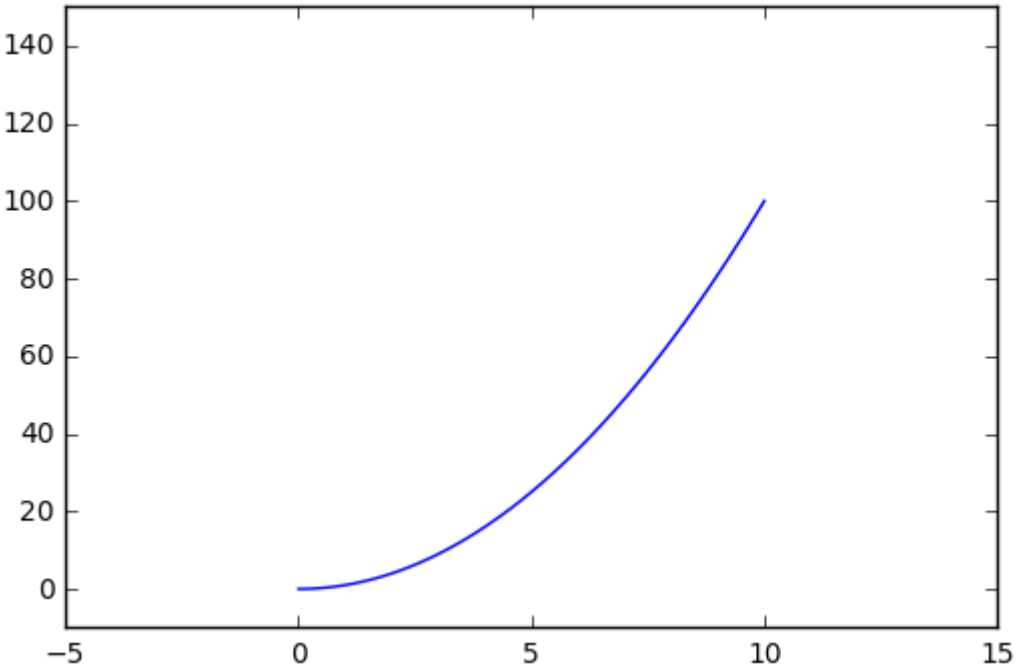
In [30]:
```
X = np.linspace(0, 10, 5) # 5 = resolution
Y = [f(x) for x in X]

fig, ax = plt.subplots()
ax.plot(X,Y)

plt.show()
```



Notice how "broken" the graph looks

With the function `ax.set_xlim` , `ax.set_ylim` we can change the boundaries of the axes:

In [32]:
```
fig, ax = plt.subplots()

X = np.linspace(0, 10, 100) # 0 = min X, 10 = max X, 100 = resolution...
Y = [f(x) for x in X]

ax.set_xlim(-5, 15)
ax.set_ylim(-10, 150)
ax.plot(X,Y)

plt.show()
```



We can also put labels on the axes and over the plot:

In [33]:
```
fig, ax = plt.subplots()

X = np.linspace(0, 10, 100) # 0 = min X, 10 = max X, 100 = resolution...
Y = [f(x) for x in X]

ax.set_xlim(-5, 15)
ax.set_ylim(-10, 150)
ax.plot(X,Y)

ax.set_xlabel("Genes")
ax.set_ylabel("Unicorns")
ax.set_title("Number of Unicorns in Genes")


plt.show()
```

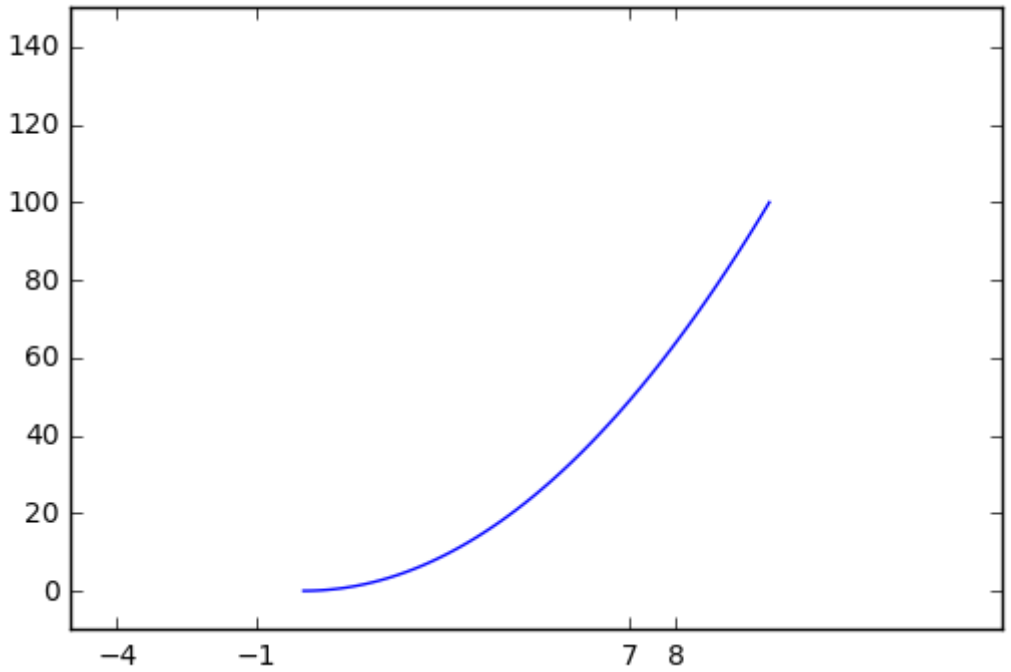
You can specify size, font and style in labels

In [36]:

```
fig, ax = plt.subplots()

X = np.linspace(0, 10, 100) # 0 = min X, 10 = max X, 100 = resolution...
Y = [f(x) for x in X]

ax.set_xlim(-5, 15)
ax.set_ylim(-10, 150)
ax.plot(X,Y)

ax.set_xlabel("Genes")
ax.set_ylabel("Unicorns")
ax.set_title("Number of Unicorns in Genes", fontname="Comic Sans MS", font

plt.show()
```

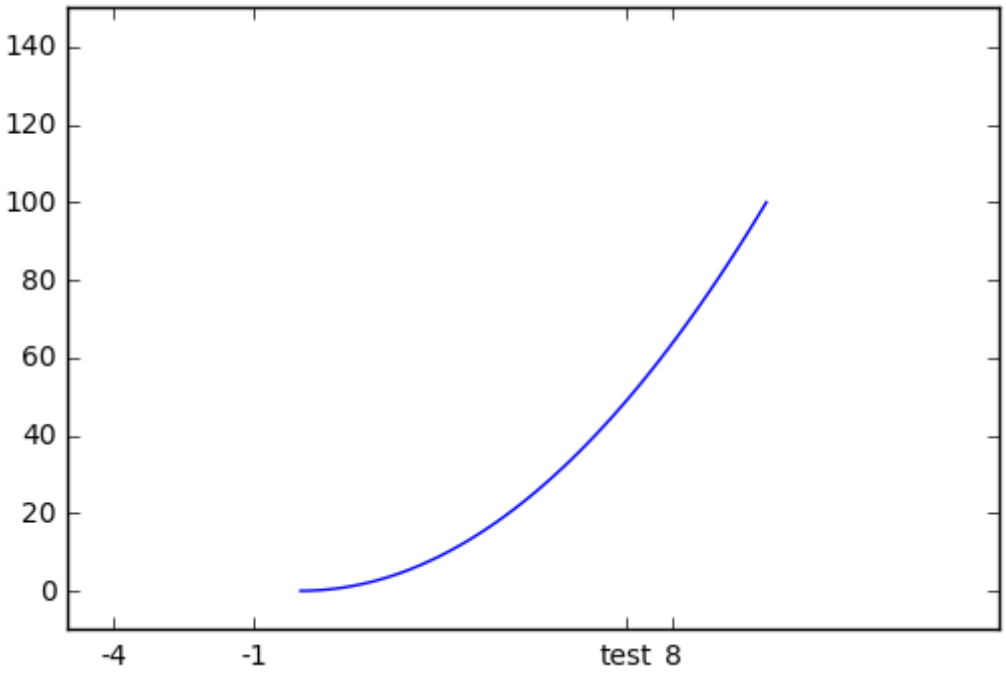
With `ax.set_xticks` you can specify which ticks will appear on each axis:

In [38]:
```
fig, ax = plt.subplots()
ax.set_xlim(-5, 15)
ax.set_ylim(-10, 150)
ax.plot(X,Y)
ax.set_xticks([-4,-1,7,8])
plt.show()
```



You can also change the tick label:

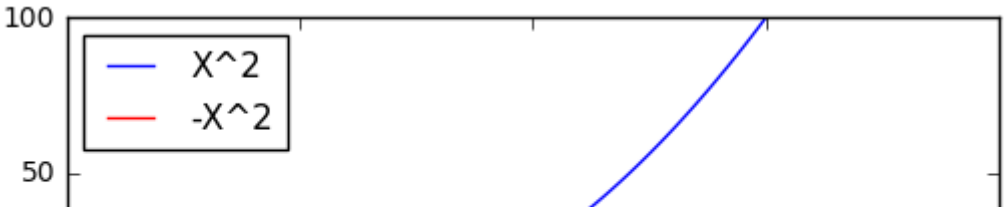
In [45]:
```
fig, ax = plt.subplots()
ax.set_xlim(-5, 15)
ax.set_ylim(-10, 150)
ax.plot(X,Y)
ax.set_xticks([-4,-1,7,8])

ticks = ax.get_xticks().tolist()
ticks[2] = "test"
ax.set_xticklabels(ticks)

plt.show()
```



The `plot` function returns a table of legends. We can add these legends to the plot with `plt.legend` :

In [62]:
```
fig, ax = plt.subplots()
ax.set_xlim(-5, 15)
ax.set_ylim(-100, 100)
legends = ax.plot(X,Y, 'b', X, [-y for y in Y], 'r')
print (legends)
plt.legend(legends, ["X^2", "-X^2"], loc=2) # loc=2 --> upper left
plt.show()
```

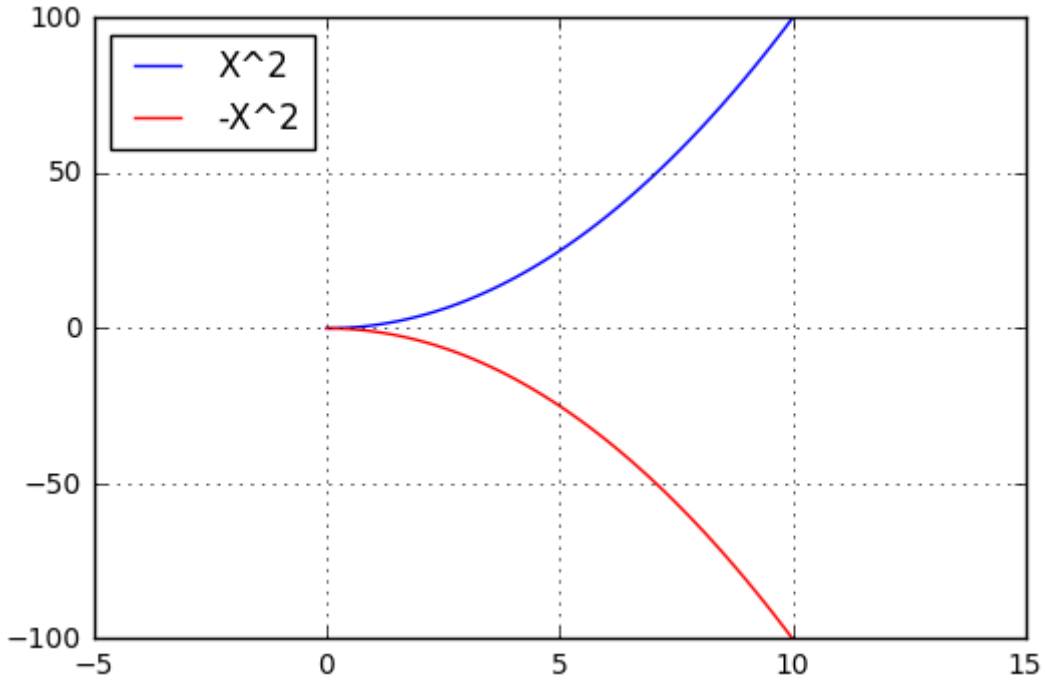
```
[<matplotlib.lines.Line2D object at 0x10e3c2208>, <matplotlib.lines.Line2D
object at 0x10df87d68>]
```



More about the loc (location of the legend) see here: http://matplotlib.org/api/legend_api.html#matplotlib.legend.Legend

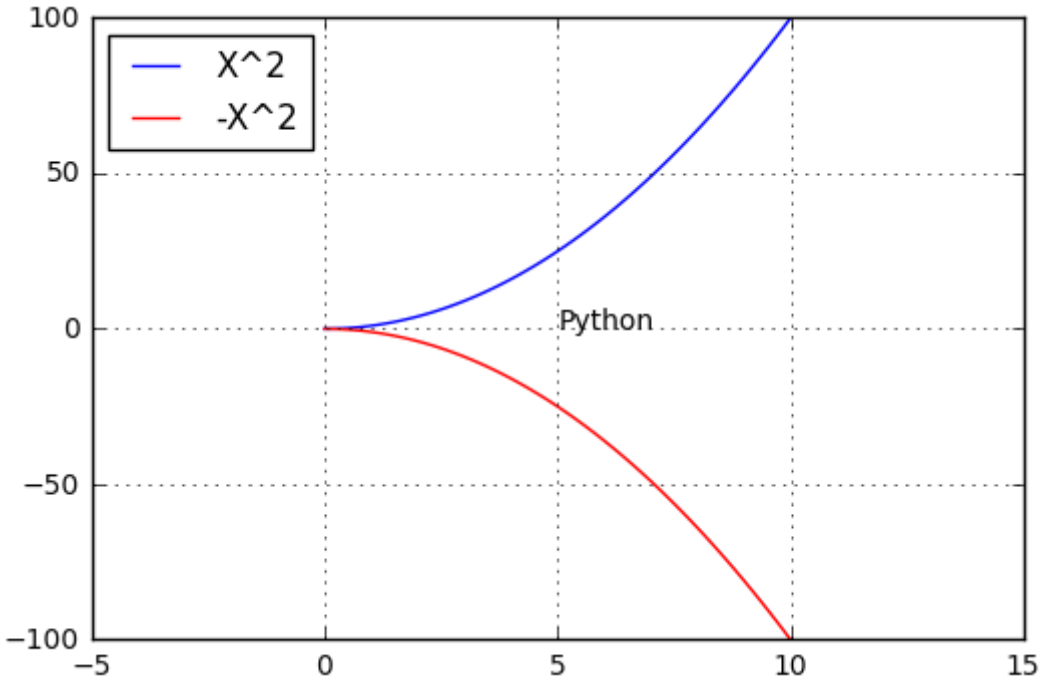
We can also add a grid:

In [64]:
```
fig, ax = plt.subplots()
ax.set_xlim(-5, 15)
ax.set_ylim(-100, 100)
legends = ax.plot(X,Y, 'b', X, [-y for y in Y], 'r')

ax.grid(True)

plt.legend(legends, ["X^2", "-X^2"], loc=2) # loc=2 shmainei panw aristera
plt.show()
```



Add a text at point X, Y:

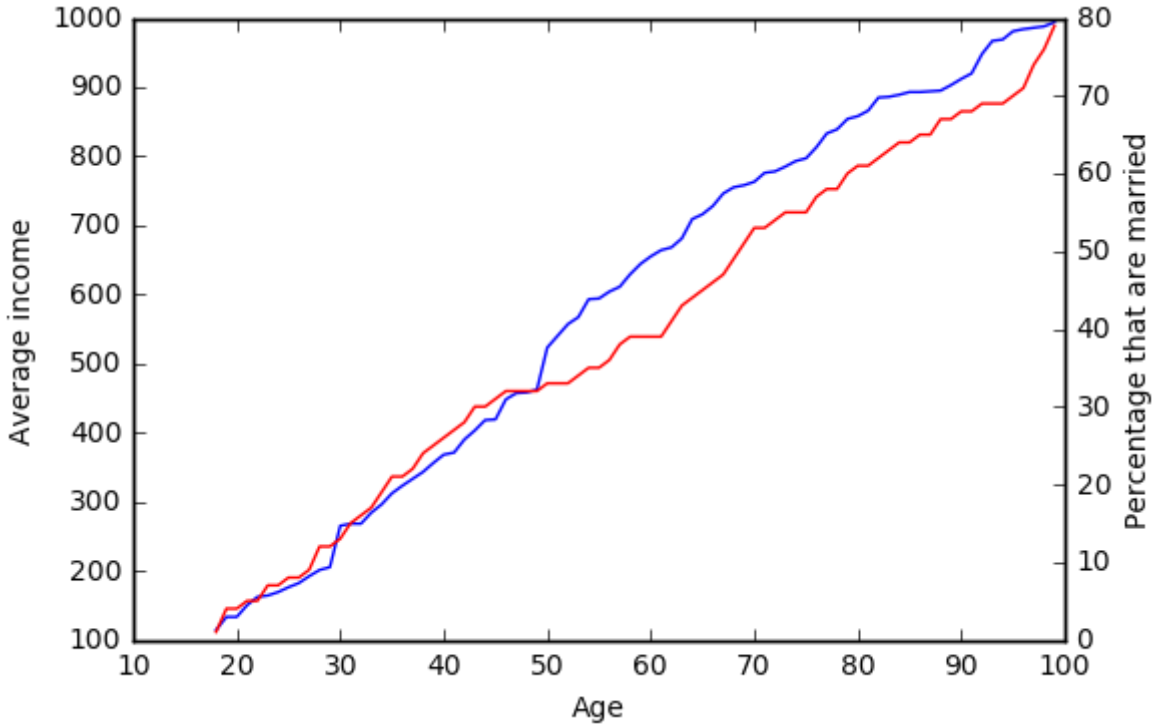
In [65]:
```
fig, ax = plt.subplots()
ax.set_xlim(-5, 15)
ax.set_ylim(-100, 100)
legends = ax.plot(X,Y, 'b', X, [-y for y in Y], 'r')
ax.grid(True)

ax.text(5,0,"Python")

plt.legend(legends, ["X^2", "-X^2"], loc=2) # loc=2 shmainei panw aristera
plt.show()
```



There is also the annotate function with which you can draw arrows:

Often, we want to print two plots that share the same axis. Suppose e.g. that we want to share the X axis:

In [70]:
```
fig, ax = plt.subplots()
import random

age = [x for x in range(18,100)]
income = sorted([random.randint(100,1000) for x in age])
percentage_married = sorted([random.randint(0, 80) for x in age])

legends_income = ax.plot(age, income, 'b')

# Create a copy of x axes
ax_new = ax.twinx()

# Plot the second graph on this copy
legends_married = ax_new.plot(age, percentage_married, 'r')

ax.set_xlabel("Age")
ax.set_ylabel("Average income")
ax_new.set_ylabel("Percentage that are married")

plt.show()
```

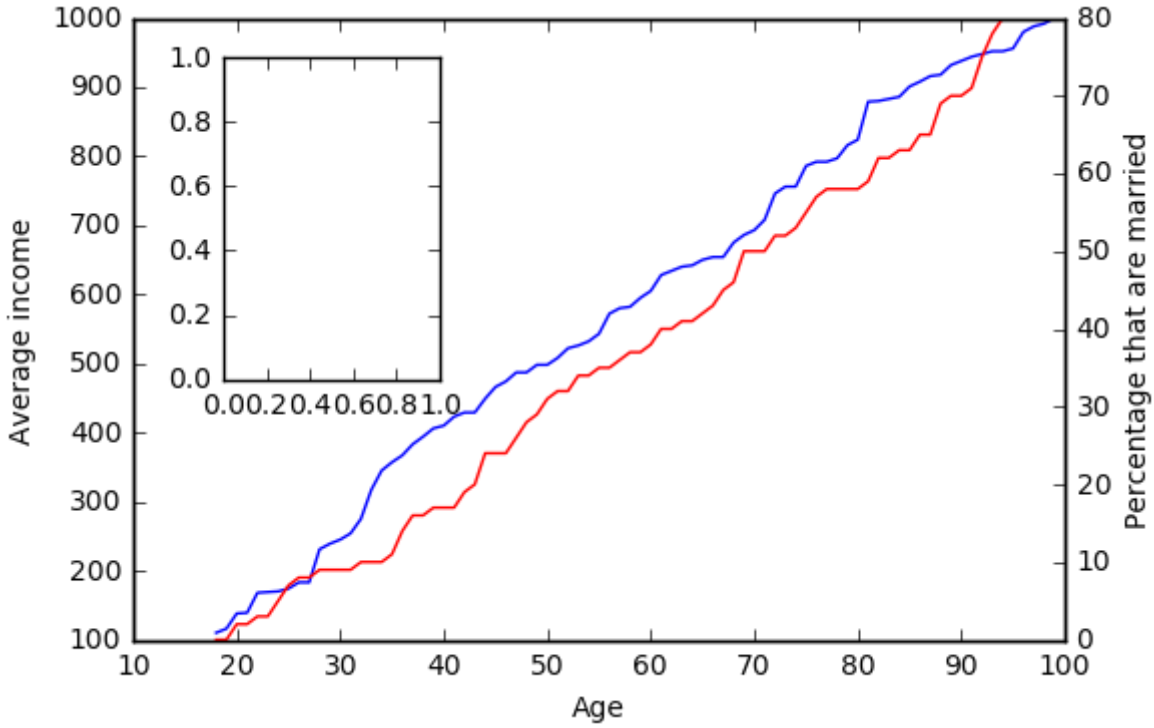


We can also embed a whole new plot inside another with the command fig.add_axes (). `add_axes` **IGNORES** the size of the axes (eg X has a size from 10 to 100 above). On the contrary, it considers that the ENTIRE plot is a Cartesian product [0,1] X [0,1]. With `add_axes` we define the dimensions of the new plot on the old one.

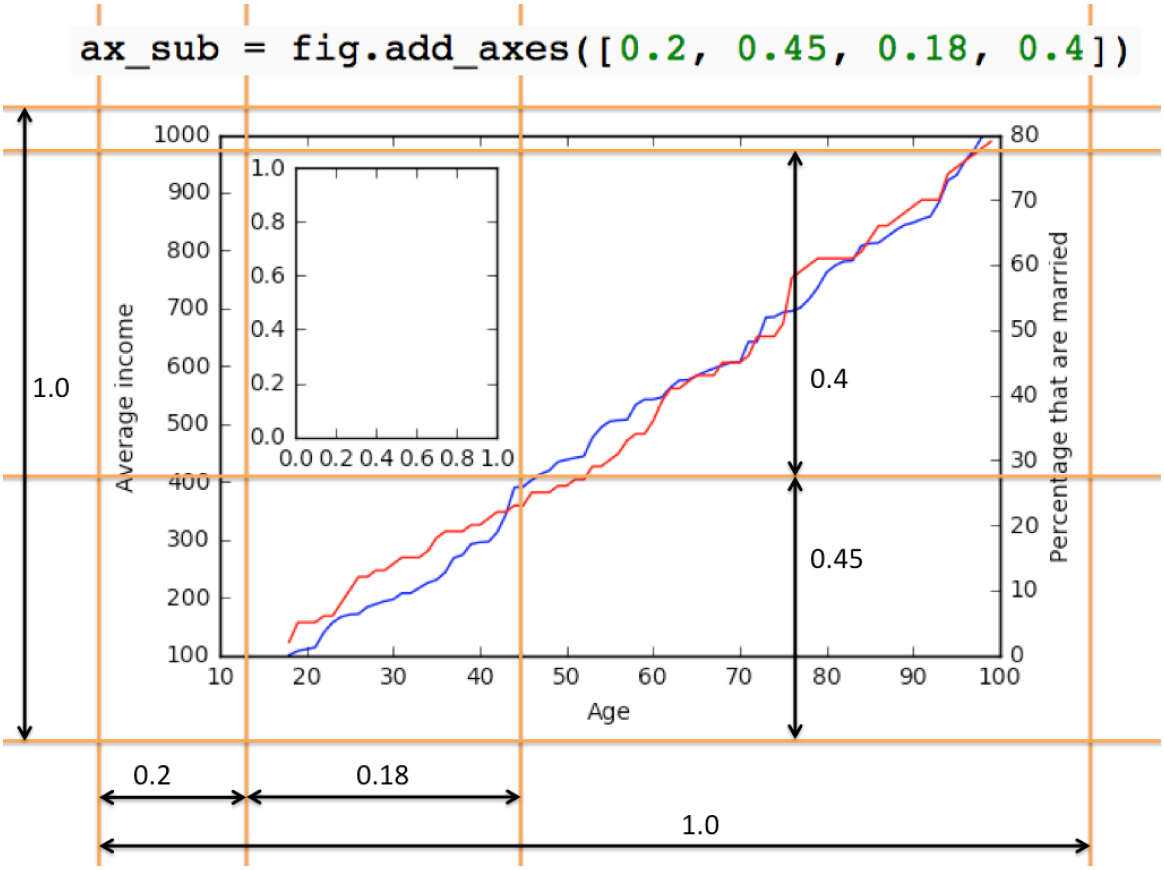
For Example:

In [82]:

```
fig, ax = plt.subplots()
import random

age = [x for x in range(18,100)]
income = sorted([random.randint(100,1000) for x in age])
percentage_married = sorted([random.randint(0, 80) for x in age])

legends_income = ax.plot(age, income, 'b')

# Create a copy of x axes
ax_new = ax.twinx()

# Plot the second graph on this copy
legends_married = ax_new.plot(age, percentage_married, 'r')

ax.set_xlabel("Age")
ax.set_ylabel("Average income")
ax_new.set_ylabel("Percentage that are married")


# Create a new subplot
ax_sub = fig.add_axes([0.2, 0.45, 0.18, 0.4])


plt.show()
```

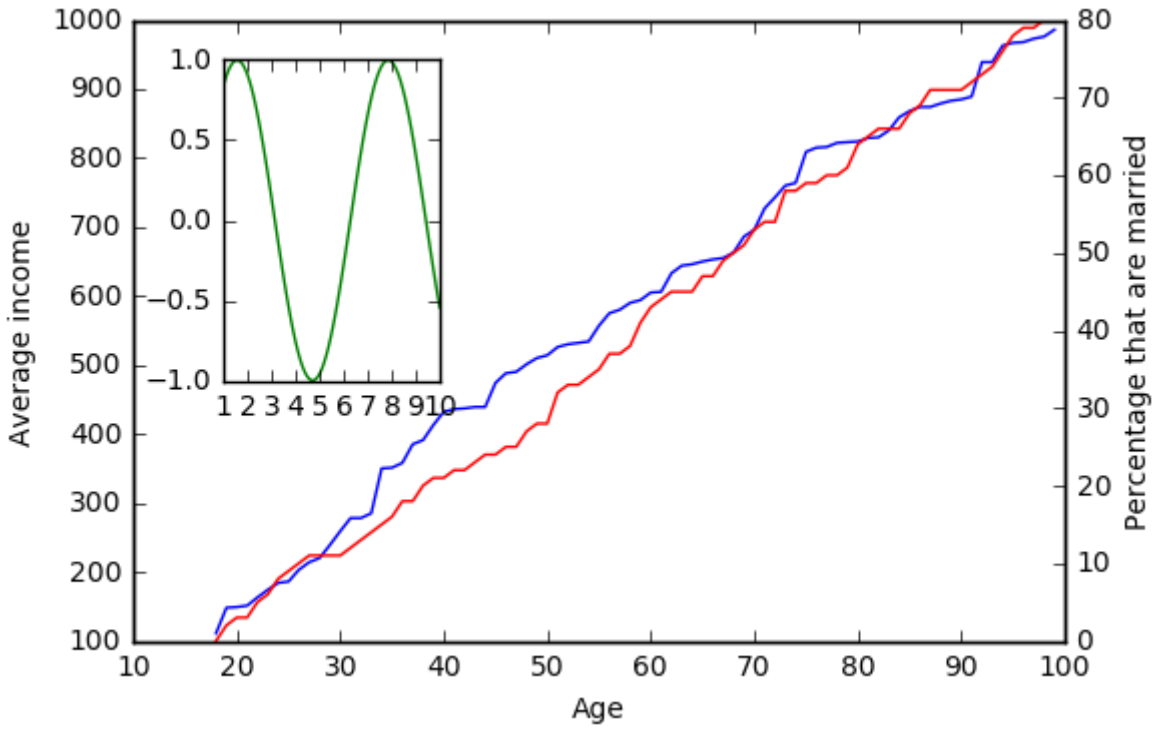


The semantics of the `add_axes` parameters are shown in the following figure:



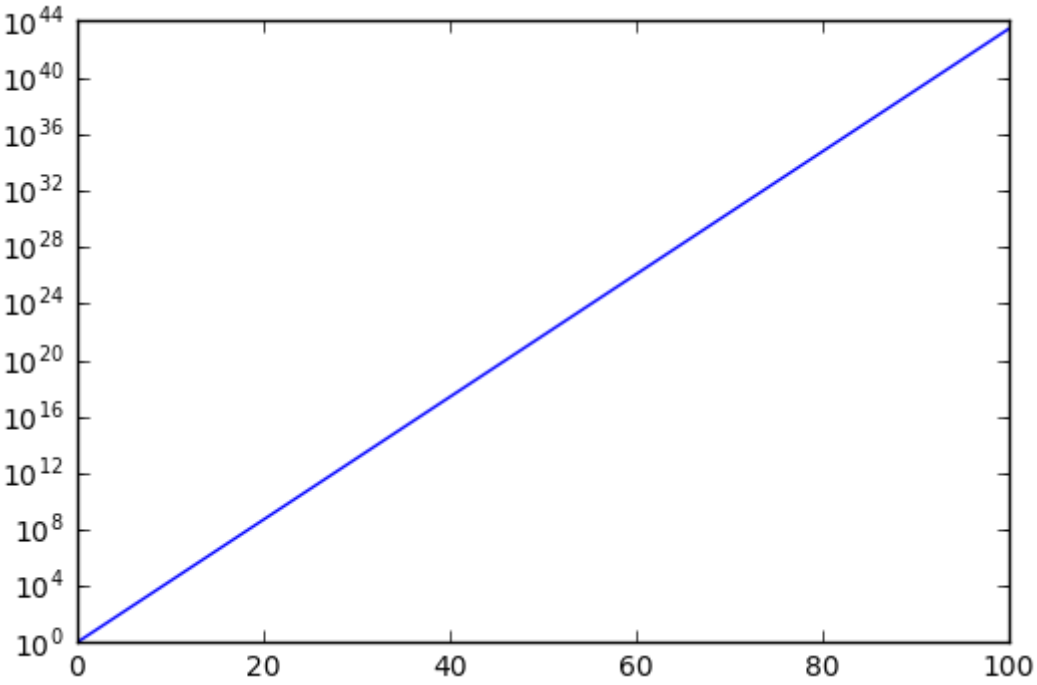
Let's plot something in the sub-plot:

In [85]:
```
fig, ax = plt.subplots()
import random

age = [x for x in range(18,100)]
income = sorted([random.randint(100,1000) for x in age])
percentage_married = sorted([random.randint(0, 80) for x in age])

legends_income = ax.plot(age, income, 'b')

# Create a copy of x axes
ax_new = ax.twinx()

# Plot the second graph on this copy
legends_married = ax_new.plot(age, percentage_married, 'r')

ax.set_xlabel("Age")
ax.set_ylabel("Average income")
ax_new.set_ylabel("Percentage that are married")


# Create a new subplot
ax_sub = fig.add_axes([0.2, 0.45, 0.18, 0.4])
sub_X = np.linspace(1,10,100)
sub_Y = np.sin(sub_X)
ax_sub.plot(sub_X,sub_Y, 'g')

plt.show()
```



We can also change the scale of the axes to be logarithmic:

In [88]:
```
fig, ax = plt.subplots()

X = np.linspace(0,100,1000)
Y = np.exp(X)

ax.plot(X,Y)

ax.set_yscale("log")
plt.show()
```



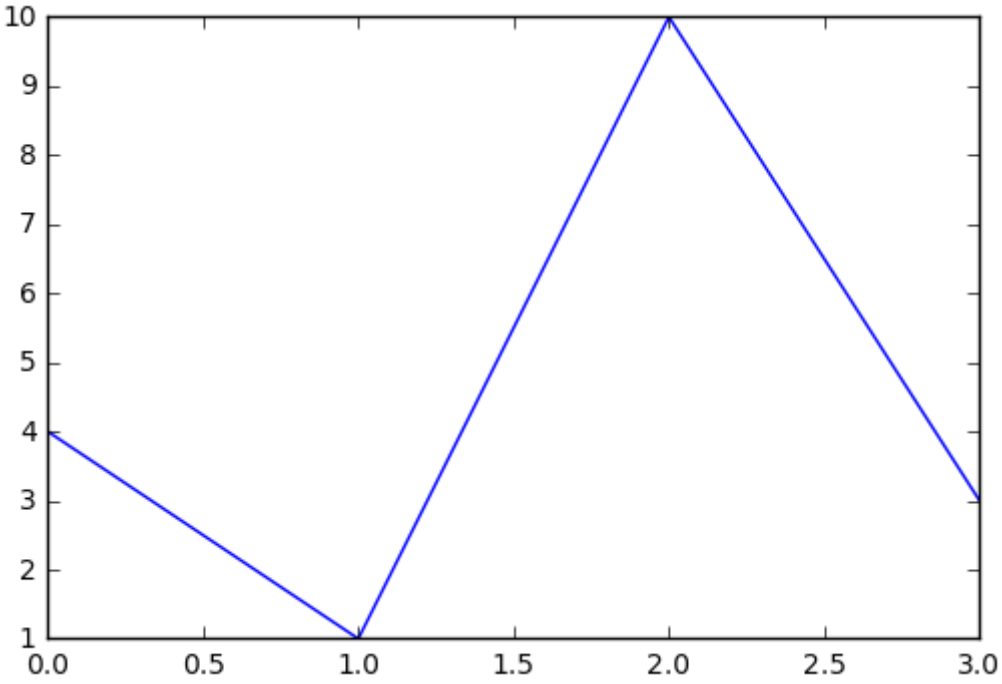
We can save a plot via `plt.savefig` . Depending on the extension in the file name, it will save it in a different format. **CAUTION!** you must call savefig BEFORE `plt.show` .

Some comments on file formats:

- In general there are two types: raster graphics and vector graphics.
- raster graphics (example: png, jpeg, bmp) are easily embeded in a paper or in a web page. The problem is that their resolution is finite. Or else you cannot zoom indefinetly without the problem of aliazing start affecting the quality of the image. To control the aliazing problem you can set the dpi (dots per inches) parameter. The higher the dpi the better the image resolution and the larger the filesize.
- Vector graphics, on the other side (example: eps, svg, pdf) cannot be easily embedded in a paper or in a web page but have infinite resolution.

In [87]:
```
fig, ax = plt.subplots()

ax.plot([4,1,10,3])

plt.savefig("figure.png")
plt.savefig("figure.jpg")
plt.savefig("figure.eps")
plt.savefig("figure.tiff")
plt.savefig("figure.pdf")

plt.show()
```

```
<matplotlib.figure.Figure at 0x10e54dd68>
```

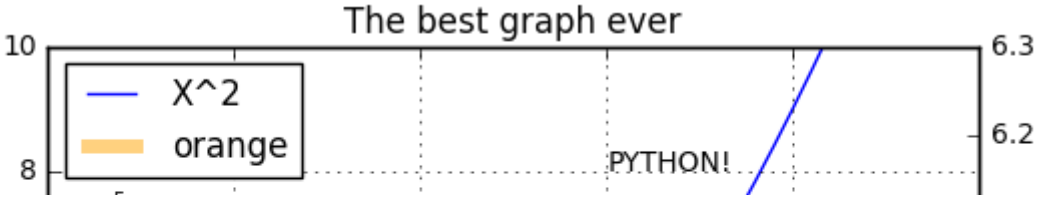


The complete example:

In [89]:

```
def x2(x):
    return x*x

fig, ax = plt.subplots()
leg_orange, = ax.plot([1,2,3], [4,6,5,], '-', color="orange", lw="5", alpha
ax.plot([1,2,3], [5,6,1,], '-', color="black", lw="5", alpha=0.5)
ax.plot([1,2,3], [5,6,1,], '*', color="black", lw="5", mew=5)
X = [x/10.0 for x in range(0,100)]
Y = [x2(x) for x in X]
leg_x2, = ax.plot(X, Y)

ticks = ax.get_xticks()
print (ticks)
#ticks = [str(x) + " a " for x in ticks]
# ax.set_xticklabels(ticks)
#ax.set_xticks([x for x in range(-1,5, 2)])
ax.set_xlabel(" GENES ")
ax.set_ylabel(" WHATEVER", fontsize=20)
ax.set_title("The best graph ever")
ax.grid(True)

ax_new = ax.twinx()
ax_new.plot([0,4], [6,6], color="green")
ax_new.set_ylabel("RIGHT LABEL")

ax.set_xlim(-1,4)
ax.set_ylim(0,10)

ax.text(2, 8, "PYTHON!")

plt.legend([leg_x2, leg_orange], ["X^2", "orange"], loc=2)
ax2 = fig.add_axes([0.2, 0.4, 0.2, 0.3])
X2 = [x/10.0 for x in range(0,100)]
Y2 = [3**x for x in X2]
ax2.set_yscale("log")

ax2.plot(X2, Y2)

plt.savefig("figure.png")
plt.savefig("figure.jpg")
plt.savefig("figure.eps")
plt.savefig("figure.tiff")
plt.show()
```

```
[  0.   2.   4.   6.   8.  10.]
```

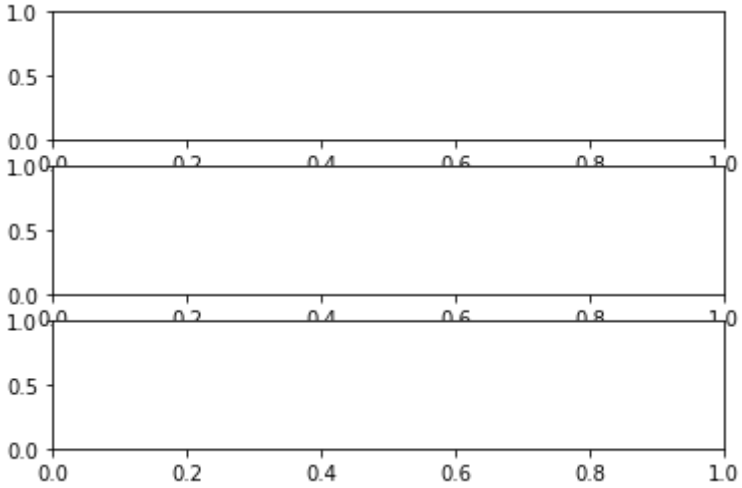


subplots

We can have miltiple plots aligned on a grid. This is possible by providing a numberof dimension in the `plt.subplots` :

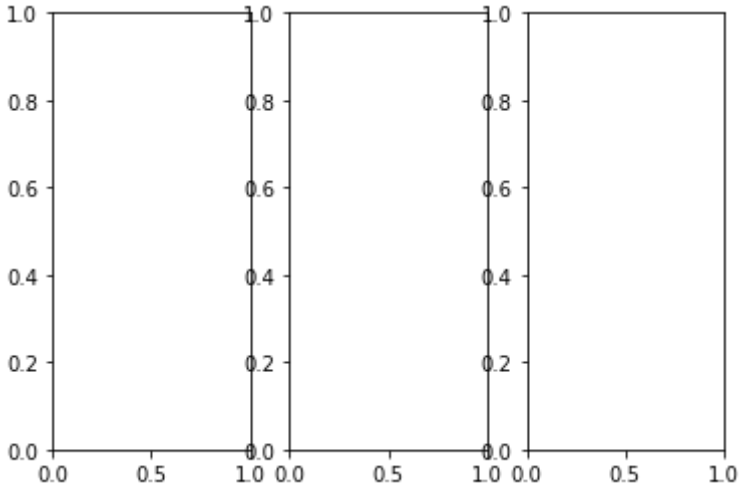
For example: 3 vertical subplots:

In [3]:
```
fig, ax = plt.subplots(3)
```



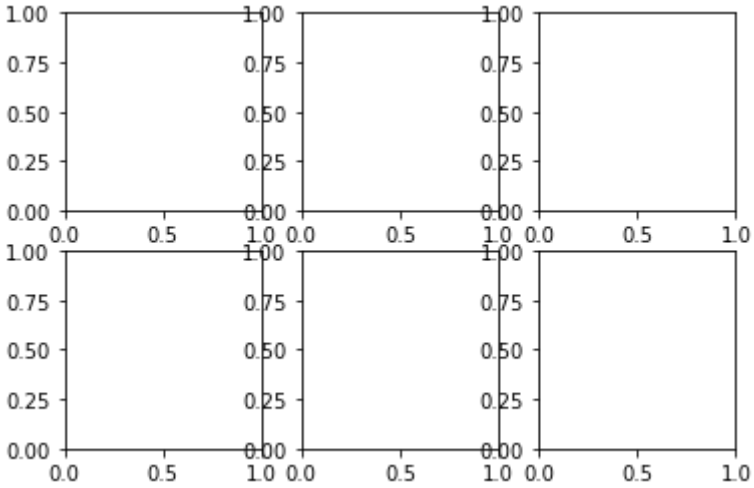
3 horizontal subplots:

In [4]:
```
fig, ax = plt.subplots(1,3)
```



A 2X3 grid of plots:

In [5]:
```
fig, ax = plt.subplots(2,3)
```



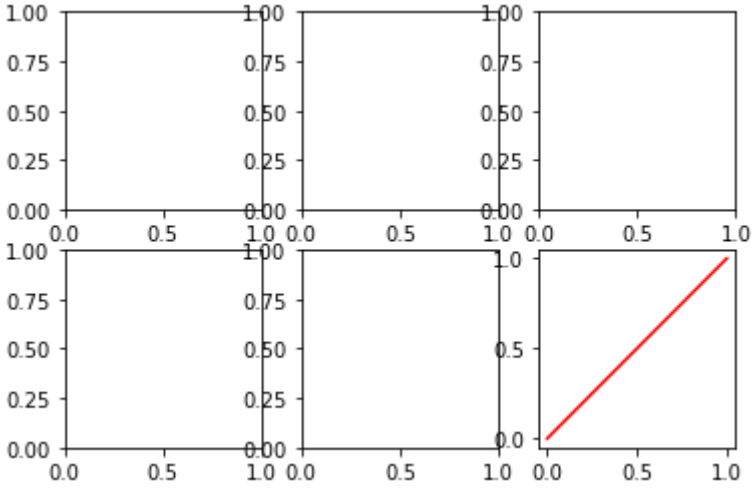
We observe that `ax` is a 2X3 array:

In [6]:
```
ax.shape
```

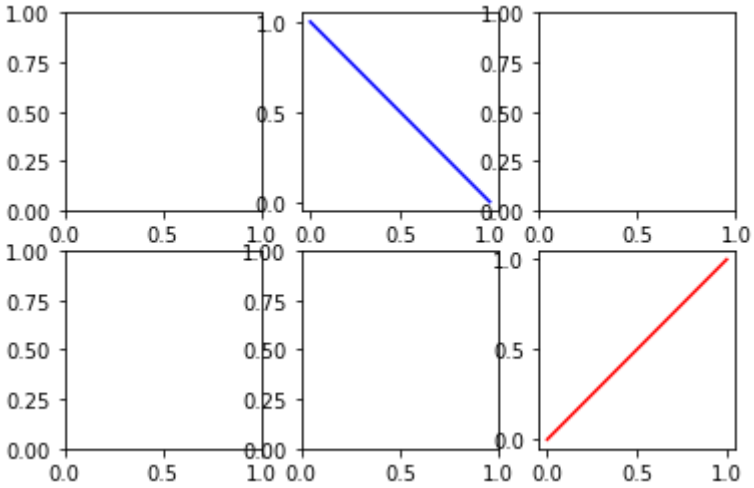
Out[6]:
```
(2, 3)
```

So we can refer to any element of this array in order to "fill" some of these sub-plots:

In [8]:
```
fig, ax = plt.subplots(2,3)
ax[1][2].plot([0,1], [0,1], 'r') # Lower right
plt.show()
```



In [9]:
```
fig, ax = plt.subplots(2,3)
ax[1][2].plot([0,1], [0,1], 'r') # Lower right
ax[0][1].plot([0,1], [1,0], 'b') # Upper middle
plt.show()
```



Some notes:

- matplotlib is huge! Check out the number of different plots it supports: http://matplotlib.org/gallery.html .
- matplotlib is not the only one. Another library is seaborn which is oriented in distribution, histogram and heatmaps plotting .
- Matplotlib has been "accused" of not producing publication ready plots. That is, aesthetically they want a little "retouch" before they get into a scientific paper. Read more:
  - https://github.com/jbmouret/matplotlib_for_papers
  - http://blog.dmcdougall.co.uk/publication-ready-the-first-time-beautiful-reproducible-plots-with-matplotlib/
  - http: //www.reddit.com/r/bioinformatics/comments/2cnv9g /polishing_your_python_matplotlib_plots_for/
  - http://nipunbatra.github.io/2014/08/latexify/

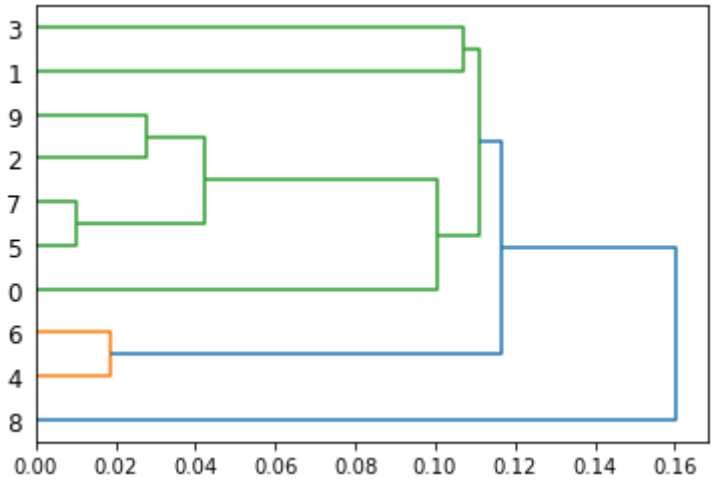
Dendrogram

Another type of plots particularly useful in phylogenetics are dendrograms. For example suppose we have 10 objects and we get all pairwise distances:

In [10]:
```
import random

o = [random.random() for x in range(10)]

def distance(a,b): # A simple distance function. The absolute difference
    return abs(a-b)
```

In [42]:
```
# Get all pairwise distances

from itertools import combinations
all_distances = [distance(x,y) for x,y in combinations(o,2)]
```

In [43]:
```
# Create a dendrogram

from scipy.cluster.hierarchy import dendrogram, linkage

l = linkage(all_distances)
dn = dendrogram(l, orientation='right')
```



In [46]:
```
# We can confirm the dendrogram by inspecting the values of the leafs.
# for example nodes 7 and 5 is the closest pair.
for i,x in enumerate(o):
    print (i,x)
```

```
0 0.535941403079773
1 0.4246640716083616
2 0.6639266980035059
3 0.3177651541741866
4 0.2011939407247051
5 0.7161232515336856
6 0.18269991970324606
7 0.7061565998207685
8 0.022555869599750866
9 0.6362676402564162
```

plotly

Matplotlib belongs in the category of non-interactive plots. This is suitable for presentations / papers but not for visual exploration and inspection!

Very good libraries for data inspection are opened such as bokeh and plot.ly.

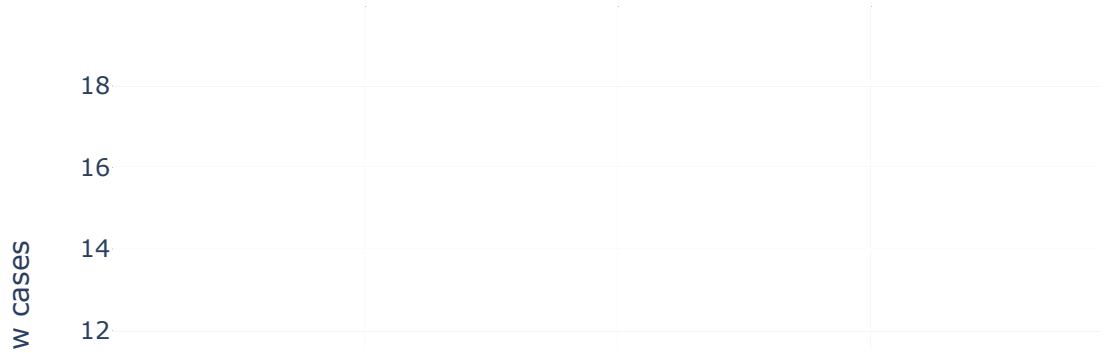
plotly needs installation:

```
!pip install plotly
```

Let's look at some examples with plotly:

A typical plot:

In [11]:
```
import plotly.express as px

fig=px.line(
    x=range(1,11),
    y=[2,4,6,7,8,9,10,15,18,19],
    labels={'x':'Day','y':'Number of new cases'})
fig.update_layout({'plot_bgcolor': '#d7dade','paper_bgcolor': '#d7dade'})
fig.show()
```



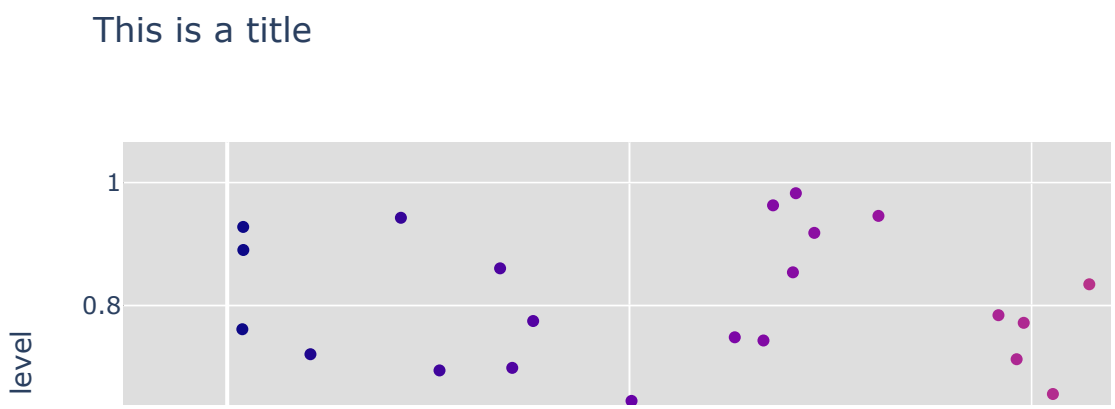
This plot is interactive. We can save it as html:

In [3]:
```
fig.write_html("file.html")
```

Scatter plot with plotly:

In [12]:

```
import numpy as np

X = np.random.random(100)
Y = np.random.random(100)

fig=px.scatter(
    x=X,
    y=Y,
    labels={'x':"3'UTR length", 'y':'mean expression level'},
    color=X,
    title='This is a title',
    #log_x=True,
)

fig.update_layout({'plot_bgcolor': '#dedede'})
fig.show()
```



An example with "circular plots". The chloroplastic DNA of Arabidopsis thaliana. Data are available here: ftp://ftp.ensemblgenomes.org/pub/plants/release-49/gff3 /arabidopsis_thaliana file: Arabidopsis_thaliana.TAIR10.49.chromosome.Pt.gff3. gz:

In [15]:

```
# Credit: Tzortzina Nouli
# Project in course: Introduction in python programming
# Biinformatics Graduate course 2020-2021
# Unversity of Crete, School of Medicine

import plotly.graph_objects as go
import numpy as np
from matplotlib import colors as mcolors
import re
from matplotlib import cm

def colors(rgb=False):
    for mp in ['Accent','tab10_r','Pastel1','Set3',]: #other options 'Paste
        for x in cm.get_cmap(mp).colors:
            yield 'rgb'+str(x) if rgb else x

with open('Arabidopsis_thaliana.TAIR10.49.chromosome.Pt.gff3','r') as f:
    s=f.read().split('\n')
    s=[i for i in s if i and i[0]!='#']
    l=s[0].split()[4]
    length=int(l)/360
    s=[(int(i.split()[3])/length,int(i.split()[4])/length,i.split()[6],re.
    groups=set(i[3][:3].lower() for i in s)
    s={k:[v for v in s if v[3][:3].lower()==k] for k in groups }



color=colors(True)
fig = go.Figure()

for a,w,h in [(1.06,1,'Strand 2'),(0.94,1,'Strand 1')]:
    fig.add_trace(go.Scatterpolar(r=[a] * 20*len(s),theta=np.linspace(1,360
                                  line_color='black', line_width=w,showlege

for a,th,txt in zip([0.8] * (int(l)//1000),np.linspace(0,360,int(l)//1000),
    if not txt%5:
        fig.add_trace(go.Scatterpolar(r=[a],theta=[th],textfont={'family':'

for group in s:
    c=next(color)
    for i in s[group]:
        if i[3]!='Unnamed' and i[4]!='exon':
            fig.add_trace(go.Scatterpolar(r=[1.06*(i[2]=='-')+0.94*(i[2]==

        ### Alternative version of plot with coding regions for mRNA,tRNA,

        #if i[3]=='Unnamed' and i[4]!='CDS':
        #    col={'mRNA':'Burlywood','tRNA':'olive','rRNA':'tomato'}
        #    fig.add_trace(go.Scatterpolar(r=[1.06*(i[2]=='-')+0.94*(i[2]==


fig.update_layout(
  polar=dict(angularaxis=dict(rotation=90,direction="counterclockwise",show

fig.show()
```

Seaborn

Seaborn is a library for "statistical graphics". That is, it provides methods for mapping distributions, histograms, clustering, etc.

Here is an example :https://seaborn.pydata.org/generated/seaborn.displot.html

In [16]:
```
import seaborn as sns
```

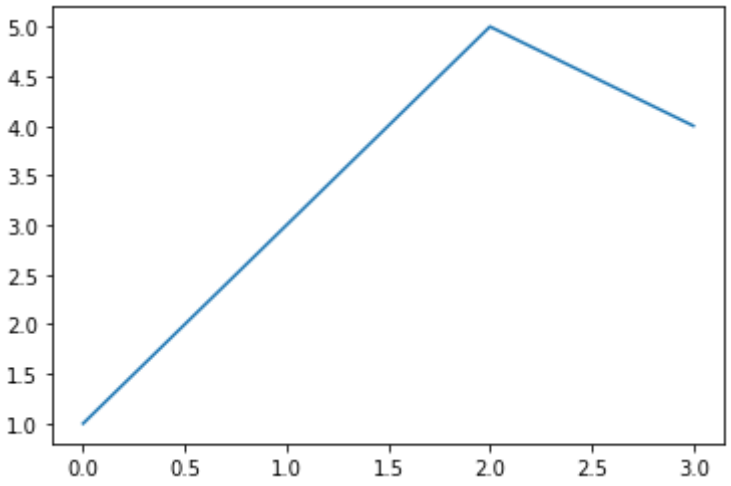
In [18]:
```
penguins = sns.load_dataset("penguins")
sns.displot(data=penguins, x="flipper_length_mm")
```

Out[18]:
```
<seaborn.axisgrid.FacetGrid at 0x7ffb595c7f40>
```

Seaborn also allows the "beautification" of matplotlib's plots. Let's look at an example:

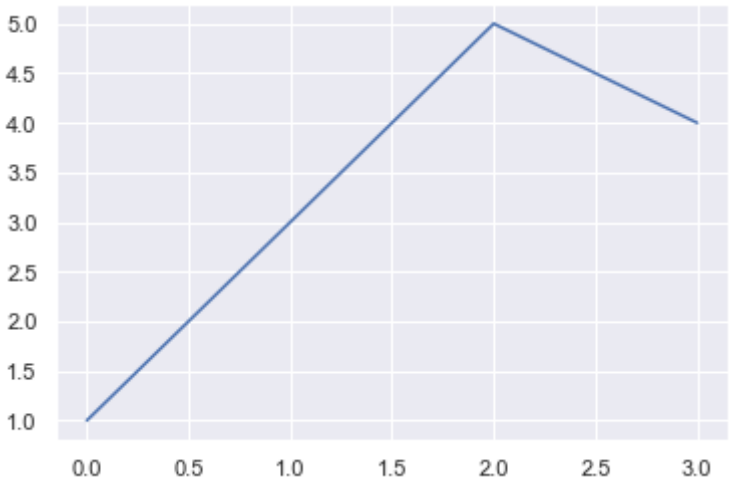
Without seaborn:

In [2]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.plot([0,1,2,3], [1,3,5,4], '-')
plt.show()
```



With seaborn:

In [3]:
```
import seaborn as sns
sns.set_theme()

fig, ax = plt.subplots()
ax.plot([0,1,2,3], [1,3,5,4], '-')
plt.show()
```



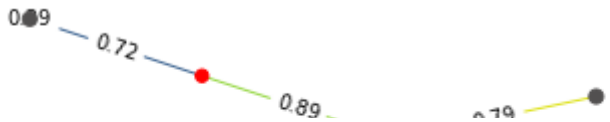
Read here: http://seaborn.pydata.org/tutorial/aesthetics.html how we can choose different "styles" for our plots from seaborn: http://seaborn.pydata.org/tutorial/aesthetics.html

Networkx

Networkx is a library for editing and visualizing graphs. After installing it (`pip install`) we can make graphs as follows:

In [37]:
```
import random

import networkx as nx

G = nx.Graph()

labels = []

# 10 random edges
for x in range(10):
    n1 = random.randint(1,10) # from a random node..
    n2 = random.randint(1,10) # ..to a random node
    c = random.random() # Random color
    G.add_edge(n1, n2,
               color=c,
               weight= '{0:.2f}'.format(random.random()), # Random edge we
              )
    labels.append('d')

pos = nx.spring_layout(G) # Get the coordinates of the nodes in 2D.

node_size = {x:x/2 for x in range(1,11)} # An abitrary calculation of node

nx.draw(
    G,
    pos,
    with_labels=False,
    node_size=[50 if i%2==0 else 40 for i in G.nodes()],
    edge_color=nx.get_edge_attributes(G,'color').values(),
    node_color=['#525050' if i%2==0 else '#FF0000' for i in G.nodes()],
)
labels = nx.get_edge_attributes(G,'weight')
nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)

#plt.figure(3,figsize=(15,10))
```

Out[37]:
```
{(9, 6): Text(-0.3246399734880249, 0.8393607649129553, '0.72'),
 (9, 1): Text(-0.13822768754437625, 0.6171603138000668, '0.89'),
 (6, 6): Text(-0.41265536497435334, 0.9446971915222568, '0.39'),
 (1, 3): Text(-0.00752602109411178, 0.2757609811871292, '0.46'),
 (1, 8): Text(0.06323129245690765, 0.5789424219216375, '0.79'),
 (3, 7): Text(0.04699044479343656, -0.12170415825792096, '0.78'),
 (4, 2): Text(0.018412541670912178, -0.796599028576672, '0.36'),
 (4, 5): Text(0.2498097374056697, -0.8219724442684326, '0.21'),
 (4, 7): Text(0.08840511218292141, -0.469289439065243, '0.73'),
 (2, 2): Text(-0.0707830023359778, -0.9492531686164785, '0.99')}
```



GeoPandas

GeoPandas is a pandas extension for visualization of geographic information.

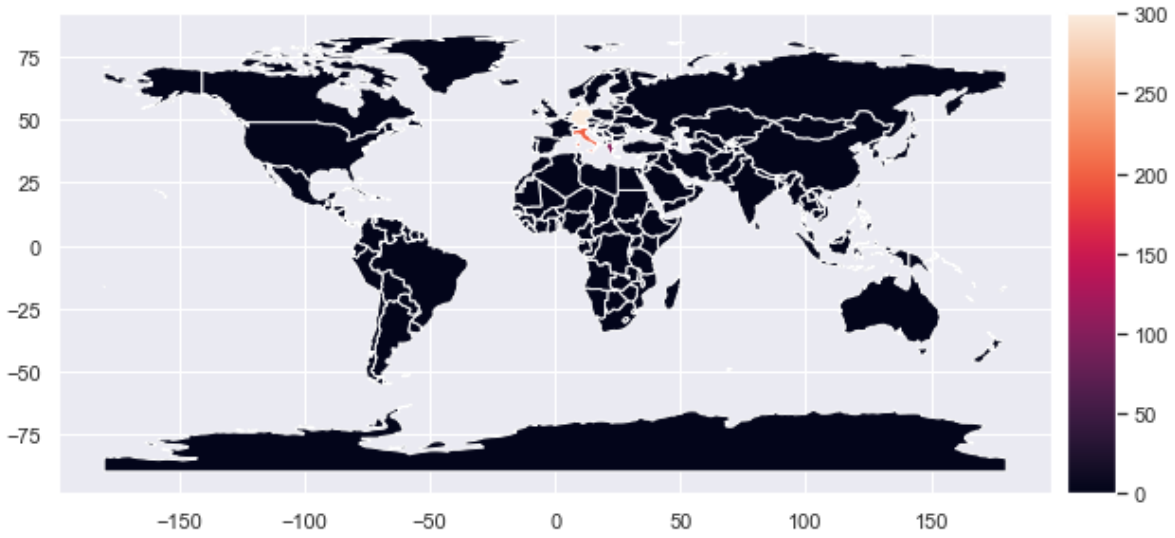
Here is an example:

In [4]:
```
import geopandas
world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowr
```

In [8]:
```
def new_function(row):
    name = row['name']
    if name == 'Greece':
        return 100
    if name == 'Italy':
        return 200
    if name == 'Germany':
        return 300

    return 0

world['new_column'] = world.apply(new_function, axis=1)
```

In [9]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, ax = plt.subplots(1, 1, figsize=(10, 8))
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.1)
world.plot(column='new_column', ax=ax, legend=True, cax=cax)
```

Out[9]: <AxesSubplot:>



plotly GEO interactive

We can also make interactive plots with geographical information through plotly:

For the following example we need the file: https://www.dropbox.com
/s/brgaopz8g2vs0ox/covid_fasta.gz?dl=1

In [11]:
```
# Credit: Tzortzina Nouli
# Project in course: Introduction in python programming
# Biinformatics Graduate course 2020-2021
# Unversity of Crete, School of Medicine

import re
import gzip
import numpy as np
from collections import defaultdict
import geopandas
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from matplotlib import cm
from matplotlib.colors import ListedColormap, LinearSegmentedColormap
import plotly.express as px

def plot_geo(file,interactive=False):
    with gzip.open(file, 'rt') as f:

        d=defaultdict(int)
        for i in f:
            s=re.findall('>hCoV-19/(.+?)/.+',i)
            if s:
                d[s[0]]+=1
    world = geopandas.read_file(geopandas.datasets.get_path('naturalearth_
    world['Covid_Count'] = world.apply(lambda row:d[row['name']], axis=1)
    if not interactive:
        fig, ax = plt.subplots(1, 1, figsize=(14, 12))
        divider = make_axes_locatable(ax)
        cax = divider.append_axes("right", size="4%", pad=0.2)
        ax=world.plot(column='Covid_Count',ax=ax,legend_kwds={'label': 'Nu
        world.boundary.plot(figsize=(14,12),edgecolor='darkgrey',ax=ax)
    else:
        fig = px.choropleth_mapbox(world, geojson=world.geometry, location
                           color_continuous_scale="inferno_r",hover_name='
                           mapbox_style="carto-positron",
                           zoom=3, center = {"lat": 37.0902, "lon": -95.71
                           opacity=0.5)

        fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
        fig.show()


file = 'covid_fasta.gz'
#plot_geo(file)

plot_geo(file,interactive=True)
```

In [ ]: