# Introduction

`matplotlib` is a Python library that creates charts.

## Pros

- Adapted to scientific publications: straight-forward charts
- Unlimited customization of charts
- Large online community

## Cons

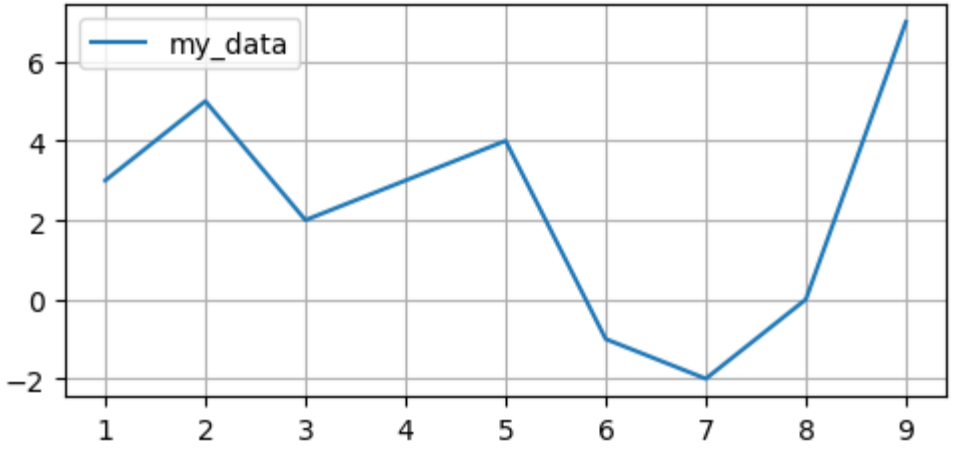
- No interactive plotting

# Simple plot

In [1]:
```
import matplotlib.pyplot as plt
```

In [2]:

```
x = range(1, 10)
y = [3, 5, 2, 3, 4, -1, -2, 0, 7]
fig, ax = plt.subplots(figsize=(5, 2.5))    # figsize is given in inches: 1 inch = 2.
ax.plot(x, y, label="my_data")
_ = ax.legend()
```



Above, a simple full blue line is used. Yet, plotting style can be fully configured when calling `plot`.

Here after, 3 lines are plotted on the same chart. The first two plots set the line style in a short format, whereas the third one use named keyword argument (`marker`, `linestyle`, `color`). Note that:

- `'x-r'` tells that:
    - marker is an 'x'
    - line must be full
    - color is red
- In the `'<-.b'` string, `-.` stands for a 'dash-dot line style'

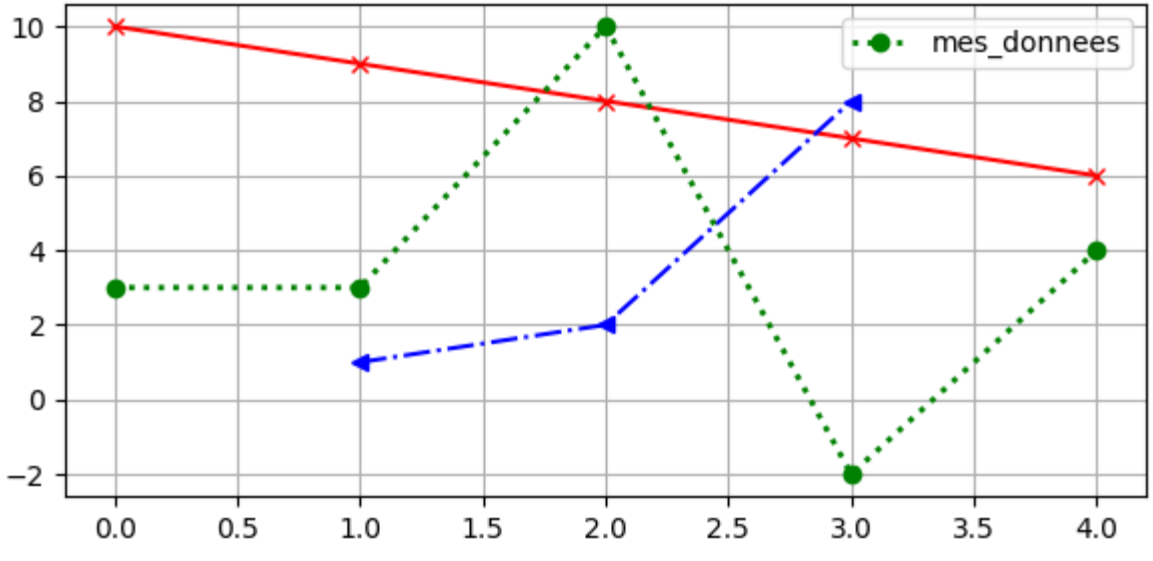
In [3]:
```
x1 = range(5)
x2 = range(1, 4)

y1 = range(10, 5, -1)
y2 = [1, 2, 8]

x3 = range(5)
y3 = [3, 3, 10, -2, 4]

fig, ax = plt.subplots(figsize=(6, 3))
ax.plot(x1, y1, 'x-r')
ax.plot(x2, y2, '<-.b')
ax.plot(x3, y3, linewidth=2, marker='o', linestyle=':', color="green", label="mes_don
ax.legend()
```

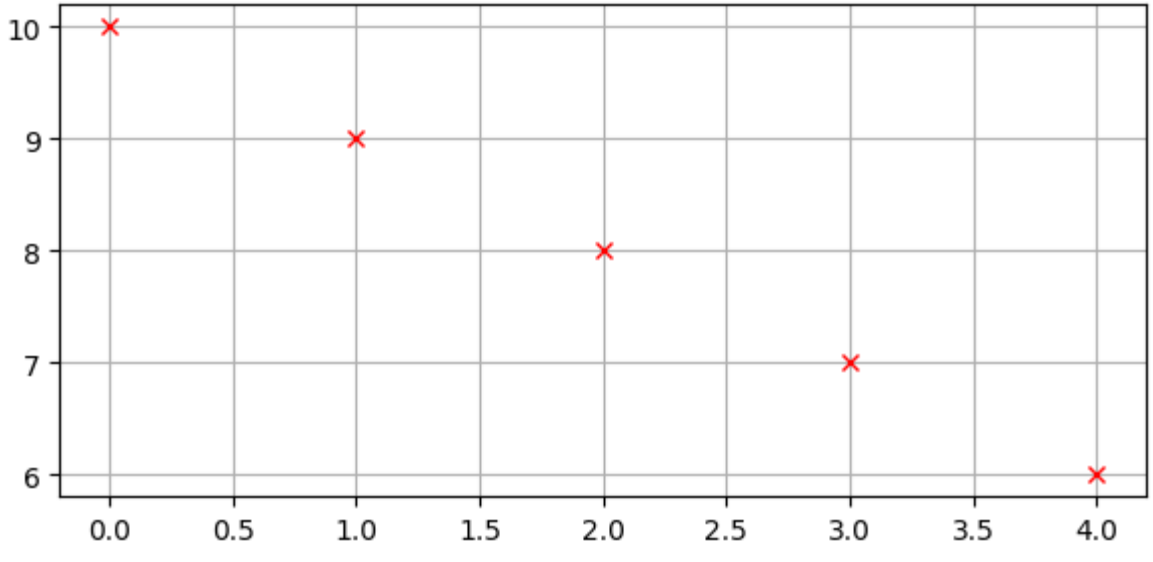
Out[3]:
```
<matplotlib.legend.Legend at 0x7f781dd2eed0>
```



Thus it is possible to plot only the markers, without any line:

In [4]:
```
fig, ax = plt.subplots(figsize=(6, 3))
ax.plot(x1, y1, 'xr')
```

Out[4]:
```
[<matplotlib.lines.Line2D at 0x7f781dd41370>]
```



# 3D plots

## Case study definition

Let's define values of a 2-variables function:

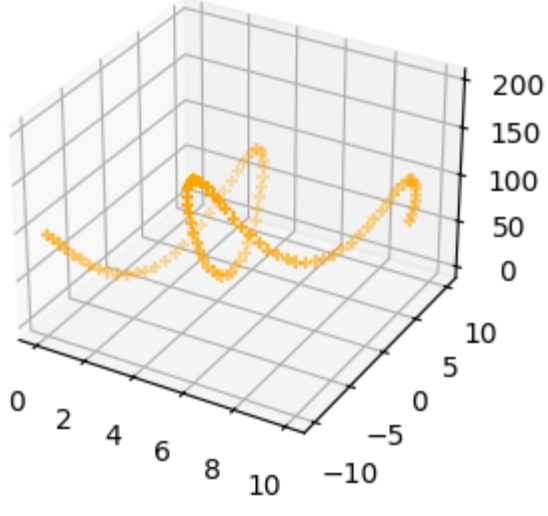
In [5]:
```
import numpy as np
x = np.linspace(0, 10, 100)
y = 10 * np.sin(np.linspace(5, 15, 100))
z = (x-y)**2 + x*y
```

## Points

The `scatter` function with a 3d projection can plot the data with only markers:

In [6]:
```
fig, ax = plt.subplots(figsize=(6, 3), subplot_kw={'projection': '3d'})
ax.scatter(x, y, z, marker='+', color='orange')
```

Out[6]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f781ffdea80>
```



## Surface

The previous example used specific values for `x` and `y` data. But if a 2-variable function must be completely described, a grid evaluation ( `np.meshgrid` ) and a surface plot ( `ax.plot_surface` ) are needed.

$f: (x, y) \rightarrow (x-5)^2 + y^2$

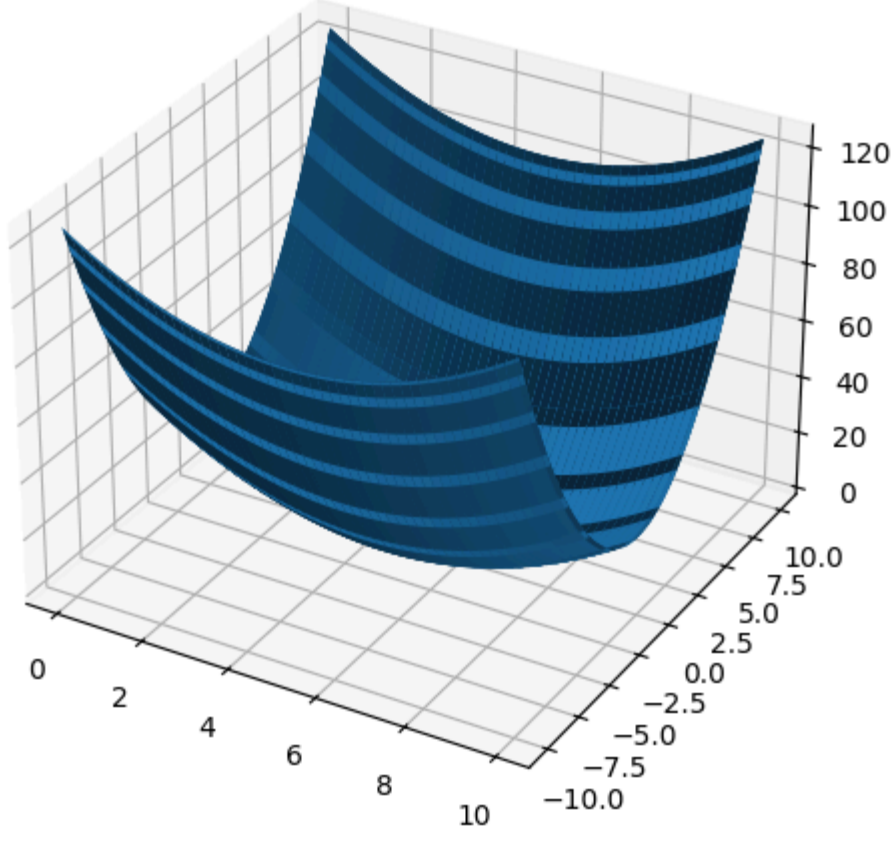
In [7]:

```
xx, yy = np.meshgrid(x, y)
zz = (xx - 5)**2 + yy**2

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(xx, yy, zz)
```

Out[7]:

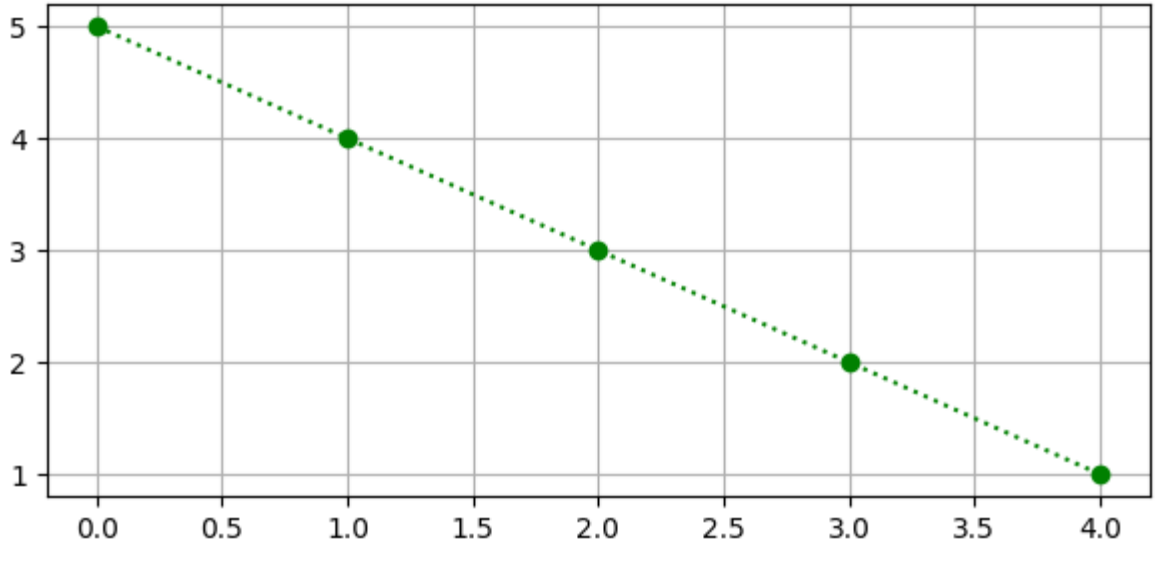
```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x7f781ffdc560>
```



# Saving a figure

When using `matplotlib`, figures are shown in an interactive way. Whenever saving to the disk is required, one must use the `savefig` method of the figure:

In [8]:
```
fig, ax = plt.subplots(figsize=(6, 3))
ax.plot(range(5), range(5, 0, -1), 'o:g')
fig.savefig('figures/super_figure.png', dpi=200, bbox_inches='tight')
```



Note the arguments:

- `bbox_inches='tight'` : use all available space in the window
- `dpi` : set the quality of the figure in **pixels per inch**.

  Most scientific journals require a dpi higher than 200. Yet, a larger dpi comes with:

  - a larger disk space
  - a longer writing time

  Hence, prefer a not to high dpi in your every-day life.