Session-16

September 19, 2019

1 Session - 16

In this session we learn to distinguish between entities in the namespace that point to same object and entities that are copies. We also learn how to make simple 2D plots.

Look at the output below and notice that the symbols a and b are two different entities. The unique identity numbers given by the function id (namely the memory location) are different.

```
In [1]: a=1.0

In [2]: b=1.0

In [3]: id(a)

Out[3]: 94055320303432

In [4]: id(b)

Out[4]: 94055320303360
```

The operator "is" checks if the two entities are identical by their reference. The operator "==" checks if the two entities are same by their value.

```
In [5]: a is b

Out[5]: False

In [6]: a == b

Out[6]: True
```

We can create two lists and see how to distinguish between a copy and an additional reference.

```
In [9]: x=list(range(0,10,1))

In [10]: x

Out[10]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The operator "=" creates an additional reference to the same list object. Modifying an element of one list will affect the other list too.

```
In [11]: y = x

In [12]: y

Out[12]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The following statement makes a copy of the list x. Notice the id numbers of these three objects to confirm which two are identical.

```
In [13]: z = x[:]

In [14]: z

Out[14]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [16]: id(x)

Out[16]: 139731731332344

In [17]: id(y)

Out[17]: 139731731332344

In [18]: id(z)

Out[18]: 139731731332920

In [19]: x is y

Out[19]: True

In [20]: x is z

Out[20]: False
```

When we modify an element of the list y, that gets changed in the list x too - as they are both references to the same list object. This does not affect the list z as it is a different object.

```
In [21]: y[5] = 3.1415

In [22]: x

Out[22]: [0, 1, 2, 3, 4, 3.1415, 6, 7, 8, 9]

In [23]: y

Out[23]: [0, 1, 2, 3, 4, 3.1415, 6, 7, 8, 9]

In [24]: z

Out[24]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The operator "==" checks only for values and answers if the objects have same value.

```
In [25]: x == y

Out[25]: True

In [26]: x == z

Out[26]: False

In [27]: z[5] = 3.1415

In [28]: z == x

Out[28]: True
```

1.1 Mapping functions to lists

We can apply a function to every element of a list using the map function.

```
In [29]: def sq(x):
             return x*x

In [30]: sq(2.5)

Out[30]: 6.25

In [31]: x = list(range(1,10,1))

In [32]: x

Out[32]: [1, 2, 3, 4, 5, 6, 7, 8, 9]

In [34]: y = map(sq, x)

In [35]: y

Out[35]: [1, 4, 9, 16, 25, 36, 49, 64, 81]
```

1.2 Anonymous functions

Using the keyword lambda, we can define anonymous functions. These are useful when we don't intend to reuse them later. Also these are crisp ways of expressing when the return value of the anonymous function is simple.

```
In [36]: (lambda a: a*a)(2.0)

Out[36]: 4.0

In [37]: sq(2.0)

Out[37]: 4.0
```

```
In [38]: type(sq)

Out[38]: function

In [39]: type(lambda a: a*a)

Out[39]: function

In [64]: y1 = list(map(lambda a: a*a+3.0*a+5.0, x))

In [65]: y1

Out[65]: [9.0, 15.0, 23.0, 33.0, 45.0, 59.0, 75.0, 93.0, 113.0]
```

1.3 Plotting lists

We can use the matplotlib to create 2D plots. If this module is not installed, you can do so using "pip install".

The statements starting with "%" symbol are called magic functions. The following statement conveys that the output of matplotlib should be displayed along with rest of the cells in this notebook itself.

```
In [66]: %matplotlib inline

In [67]: import matplotlib.pyplot as plt
```

Inspect the class plt to see what are the various functions it provides. The function we are interested in is "plot".

```
In [46]: dir(plt)

Out[46]: ['Annotation',
          'Arrow',
          'Artist',
          'AutoLocator',
          'Axes',
          'Button',
          'Circle',
          'Figure',
          'FigureCanvasBase',
          'FixedFormatter',
          'FixedLocator',
          'FormatStrFormatter',
          'Formatter',
          'FuncFormatter',
          'GridSpec',
          'IndexLocator',
          'Line2D',
          'LinearLocator',
          'Locator',
          'LogFormatter',
          'LogFormatterExponent',
          'LogFormatterMathtext',
          'LogLocator',
          'MaxNLocator',
          'MultipleLocator',
          'Normalize',
          'NullFormatter',
          'NullLocator',
          'PolarAxes',
          'Polygon',
          'Rectangle',
          'ScalarFormatter',
          'Slider',
          'Subplot',
          'SubplotTool',
          'Text',
          'TickHelper',
          'Widget',
          '_INSTALL_FIG_OBSERVER',
          '_IP_REGISTERED',
          '__builtins__',
          '__doc__',
          '__file__',
          '__name__',
          '__package__',
          '_auto_draw_if_interactive',
          '_autogen_docstring',
          '_backend_mod',
          '_backend_selection',
          '_hold_msg',
          '_imread',
          '_imsave',
          '_interactive_bk',
          '_pylab_helpers',
          '_setp',
          '_setup_pyplot_info_docstrings',
          '_show',
          '_string_to_bool',
          'absolute_import',
          'acorr',
          'angle_spectrum',
          'annotate',
          'arrow',
          'autoscale',
          'autumn',
          'axes',
          'axhline',
           'axhspan',
           'axis',
           'axvline',
           'axvspan',
           'bar',
           'barbs',
           'barh',
           'bone',
           'box',
           'boxplot',
           'broken_barh',
           'cla',
           'clabel',
           'clf',
           'clim',
           'close',
           'cm',
           'cohere',
           'colorbar',
           'colormaps',
           'colors',
           'connect',
           'contour',
           'contourf',
           'cool',
           'copper',
           'csd',
           'cycler',
           'dedent',
           'delaxes',
           'deprecated',
           'disconnect',
           'division',
           'docstring',
           'draw',
           'draw_all',
           'draw_if_interactive',
           'errorbar',
           'eventplot',
           'figaspect',
           'figimage',
           'figlegend',
           'fignum_exists',
           'figtext',
           'figure',
           'fill',
           'fill_between',
           'fill_betweenx',
           'findobj',
           'flag',
           'gca',
           'gcf',
           'gci',
           'get',
           'get_backend',
           'get_cmap',
           'get_current_fig_manager',
           'get_figlabels',
           'get_fignums',
           'get_plot_commands',
           'get_scale_docs',
           'get_scale_names',
           'getp',
           'ginput',
           'gray',
           'grid',
           'hexbin',
           'hist',
           'hist2d',
           'hlines',
           'hold',
           'hot',
           'hsv',
           'imread',
           'imsave',
           'imshow',
           'inferno',
           'install_repl_displayhook',
           'interactive',
           'ioff',
           'ion',
           'is_numlike',
           'is_string_like',
           'ishold',
           'isinteractive',
           'jet',
           'legend',
           'locator_params',
           'loglog',
           'magma',
           'magnitude_spectrum',
           'margins',
           'matplotlib',
           'matshow',
           'minorticks_off',
           'minorticks_on',
            'mlab',
            'new_figure_manager',
            'nipy_spectral',
            'np',
            'over',
            'pause',
            'pcolor',
            'pcolormesh',
            'phase_spectrum',
            'pie',
            'pink',
            'plasma',
            'plot',
            'plot_date',
            'plotfile',
            'plotting',
            'polar',
            'print_function',
            'prism',
            'psd',
            'pylab_setup',
            'quiver',
            'quiverkey',
            'rc',
            'rcParams',
            'rcParamsDefault',
            'rc_context',
            'rcdefaults',
            'register_cmap',
            'rgrids',
            'savefig',
            'sca',
            'scatter',
            'sci',
            'semilogx',
            'semilogy',
            'set_cmap',
            'setp',
            'show',
            'silent_list',
            'six',
            'specgram',
            'spectral',
            'spring',
            'spy',
            'stackplot',
            'stem',
            'step',
                     'streamplot',
                     'style',
                     'subplot',
                     'subplot2grid',
                     'subplot_tool',
                     'subplots',
                     'subplots_adjust',
                     'summer',
                     'suptitle',
                     'switch_backend',
                     'sys',
                     'table',
                     'text',
                     'thetagrids',
                     'tick_params',
                     'ticklabel_format',
                     'tight_layout',
                     'title',
                     'tricontour',
                     'tricontourf',
                     'tripcolor',
                     'triplot',
                     'twinx',
                     'twiny',
                     'types',
                     'unicode_literals',
                     'uninstall_repl_displayhook',
                     'violinplot',
                     'viridis',
                     'vlines',
                     'waitforbuttonpress',
                     'warnings',
                     'winter',
                     'xcorr',
                     'xkcd',
                     'xlabel',
                     'xlim',
                     'xscale',
                     'xticks',
                     'ylabel',
                     'ylim',
                     'yscale',
                     'yticks']
```

```
In [47]: dir(plt.plot)
```

```
Out[47]: ['__call__',
          '__class__',
          '__closure__',
          '__code__',
          '__defaults__',
          '__delattr__',
          '__dict__',
          '__doc__',
          '__format__',
          '__get__',
          '__getattribute__',
          '__globals__',
          '__hash__',
          '__init__',
          '__module__',
          '__name__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          'func_closure',
          'func_code',
          'func_defaults',
          'func_dict',
          'func_doc',
          'func_globals',
          'func_name']
```

```
In [48]: help(plt.plot)
```

```
Help on function plot in module matplotlib.pyplot:

plot(*args, **kwargs)
    Plot lines and/or markers to the
    :class:`~matplotlib.axes.Axes`.  *args* is a variable length
    argument, allowing for multiple *x*, *y* pairs with an
    optional format string.  For example, each of the following is
    legal::

        plot(x, y)        # plot x and y using default line style and color
        plot(x, y, 'bo')  # plot x and y using blue circle markers
        plot(y)           # plot y using x as index array 0..N-1
        plot(y, 'r+')     # ditto, but with red plusses

    If *x* and/or *y* is 2-dimensional, then the corresponding columns
    will be plotted.
```

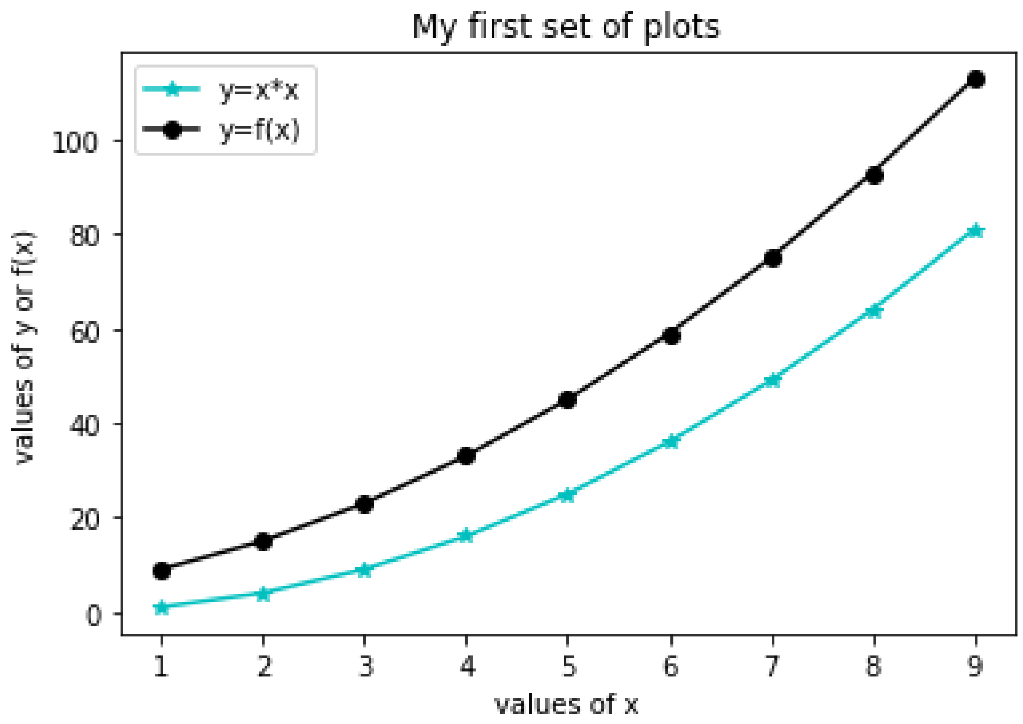
````
If used with labeled data, make sure that the color spec is not
included as an element in data, as otherwise the last case
``plot("v","r", data={"v":..., "r":...)``
can be interpreted as the first case which would do ``plot(v, r)``
using the default line style and color.

If not used with labeled data (i.e., without a data argument),
an arbitrary number of *x*, *y*, *fmt* groups can be specified, as in::

    a.plot(x1, y1, 'g^', x2, y2, 'g-')

Return value is a list of lines that were added.

By default, each line is assigned a different style specified by a
'style cycle'.  To change this behavior, you can edit the
axes.prop_cycle rcParam.

The following format string characters are accepted to control
the line style or marker:

================    ===============================
character           description
================    ===============================
``'-'``             solid line style
``'--'``            dashed line style
``'-.'``            dash-dot line style
``':'``             dotted line style
``'.'``             point marker
``','``             pixel marker
``'o'``             circle marker
``'v'``             triangle_down marker
``'^'``             triangle_up marker
``'<'``             triangle_left marker
``'>'``             triangle_right marker
``'1'``             tri_down marker
``'2'``             tri_up marker
``'3'``             tri_left marker
``'4'``             tri_right marker
``'s'``             square marker
``'p'``             pentagon marker
``'*'``             star marker
``'h'``             hexagon1 marker
``'H'``             hexagon2 marker
``'+'``             plus marker
``'x'``             x marker
``'D'``             diamond marker
``'d'``             thin_diamond marker
````

```
``'|'``             vline marker
``'_'``             hline marker
================    ===============================


The following color abbreviations are supported:

==========  ========
character   color
==========  ========
'b'         blue
'g'         green
'r'         red
'c'         cyan
'm'         magenta
'y'         yellow
'k'         black
'w'         white
==========  ========

In addition, you can specify colors in many weird and
wonderful ways, including full names (``'green'``), hex
strings (``'#008000'``), RGB or RGBA tuples (``(0,1,0,1)``) or
grayscale intensities as a string (``'0.8'``).  Of these, the
string specifications can be used in place of a ``fmt`` group,
but the tuple forms can be used only as ``kwargs``.

Line styles and colors are combined in a single format string, as in
``'bo'`` for blue circles.

The *kwargs* can be used to set line properties (any property that has
a ``set_*`` method).  You can use this to set a line label (for auto
legends), linewidth, anitialising, marker face color, etc.  Here is an
example::

    plot([1,2,3], [1,2,3], 'go-', label='line 1', linewidth=2)
    plot([1,2,3], [1,4,9], 'rs',  label='line 2')
    axis([0, 4, 0, 10])
    legend()

If you make multiple lines with one plot command, the kwargs
apply to all those lines, e.g.::

    plot(x1, y1, x2, y2, antialiased=False)

Neither line will be antialiased.

You do not need to use format strings, which are just
```

```
abbreviations.  All of the line properties can be controlled
by keyword arguments.  For example, you can set the color,
marker, linestyle, and markercolor with::

    plot(x, y, color='green', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=12).

See :class:`~matplotlib.lines.Line2D` for details.

The kwargs are :class:`~matplotlib.lines.Line2D` properties:

  agg_filter: unknown
  alpha: float (0.0 transparent through 1.0 opaque)
  animated: [True | False]
  antialiased or aa: [True | False]
  axes: an :class:`~matplotlib.axes.Axes` instance
  clip_box: a :class:`matplotlib.transforms.Bbox` instance
  clip_on: [True | False]
  clip_path: [ (:class:`~matplotlib.path.Path`, :class:`~matplotlib.transforms.Transform`)
  color or c: any matplotlib color
  contains: a callable function
  dash_capstyle: ['butt' | 'round' | 'projecting']
  dash_joinstyle: ['miter' | 'round' | 'bevel']
  dashes: sequence of on/off ink in points
  drawstyle: ['default' | 'steps' | 'steps-pre' | 'steps-mid' | 'steps-post']
  figure: a :class:`matplotlib.figure.Figure` instance
  fillstyle: ['full' | 'left' | 'right' | 'bottom' | 'top' | 'none']
  gid: an id string
  label: string or anything printable with '%s' conversion.
  linestyle or ls: ['solid' | 'dashed', 'dashdot', 'dotted' | (offset, on-off-dash-seq) |
  linewidth or lw: float value in points
  marker: :mod:`A valid marker style <matplotlib.markers>`
  markeredgecolor or mec: any matplotlib color
  markeredgewidth or mew: float value in points
  markerfacecolor or mfc: any matplotlib color
  markerfacecoloralt or mfcalt: any matplotlib color
  markersize or ms: float
  markevery: [None | int | length-2 tuple of int | slice | list/array of int | float | len
  path_effects: unknown
  picker: float distance in points or callable pick function ``fn(artist, event)``
  pickradius: float distance in points
  rasterized: [True | False | None]
  sketch_params: unknown
  snap: unknown
  solid_capstyle: ['butt' | 'round' |  'projecting']
  solid_joinstyle: ['miter' | 'round' | 'bevel']
  transform: a :class:`matplotlib.transforms.Transform` instance
  url: a url string
  visible: [True | False]
  xdata: 1D array
  ydata: 1D array
  zorder: any number

kwargs *scalex* and *scaley*, if defined, are passed on to
:meth:`~matplotlib.axes.Axes.autoscale_view` to determine
whether the *x* and *y* axes are autoscaled; the default is
*True*.

.. note::
    In addition to the above described arguments, this function can take a
    **data** keyword argument. If such a **data** argument is given, the
    following arguments are replaced by **data[<arg>]**:

    * All arguments with the following names: 'x', 'y'.
```

We now crate a scatter plot. Look at the description of the plot function above to know the syntax.

```
In [81]: p1 = plt.plot(x,y,'c*-',label='y=x*x')
         p2 = plt.plot(x,y1,'ko-',label='y=f(x)')
         plt.legend()
         plt.xlabel('values of x')
         plt.ylabel('values of y or f(x)')
         plt.title('My first set of plots')

Out[81]: <matplotlib.text.Text at 0x7f15a8638690>
```



It is important that the two lists supplied to the plot function are of the same length. You can now go ahead and create plots of different functions to explore more.

```
In [ ]:
```