Section 7 Matplotlib and Image Processing

Matplotlib

Matplotlib serves as the package to produce publication-quality figures in Python, and provides interface closely resembling to matlab.

In [1]:
```
import matplotlib as mpl # import whole package
import matplotlib.pyplot as plt # or just import submodule pylot, providing matlab-like
# these are "standard shorthands", though some poeple use other nicknames
```

In [ ]:
```
dir(mpl)
```

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

Of course you can explore the Github to see the source codes if you like.

In [4]:
```
help(plt.plot)
```

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

plot(*args, scalex=True, scaley=True, data=None, **kwargs)
    Plot y versus x as lines and/or markers.

    Call signatures::

        plot([x], y, [fmt], *, data=None, **kwargs)
        plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)

    The coordinates of the points or line nodes are given by *x*, *y*.

    The optional parameter *fmt* is a convenient way for defining basic
    formatting like color, marker and linestyle. It's a shortcut string
    notation described in the *Notes* section below.

    >>> 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

    You can use `.Line2D` properties as keyword arguments for more
    control on the appearance. Line properties and *fmt* can be mixed.
    The following two calls yield identical results:

    >>> plot(x, y, 'go--', linewidth=2, markersize=12)
    >>> plot(x, y, color='green', marker='o', linestyle='dashed',
    ...      linewidth=2, markersize=12)

    When conflicting with *fmt*, keyword arguments take precedence.


    **Plotting labelled data**
```

There's a convenient way for plotting objects with labelled data (i.e.
data that can be accessed by index ``obj['y']``). Instead of giving
the data in *x* and *y*, you can provide the object in the *data*
parameter and just give the labels for *x* and *y*::

```
>>> plot('xlabel', 'ylabel', data=obj)
```

All indexable objects are supported. This could e.g. be a `dict`, a
`pandas.DataFrame` or a structured numpy array.

Plotting multiple sets of data

There are various ways to plot multiple sets of data.

- The most straight forward way is just to call `plot` multiple times.
  Example:

  ```
  >>> plot(x1, y1, 'bo')
  >>> plot(x2, y2, 'go')
  ```

- Alternatively, if your data is already a 2d array, you can pass it
  directly to *x*, *y*. A separate data set will be drawn for every
  column.

  Example: an array ``a`` where the first column represents the *x*
  values and the other columns are the *y* columns::

  ```
  >>> plot(a[0], a[1:])
  ```

- The third way is to specify multiple sets of *[x]*, *y*, *[fmt]*
  groups::

  ```
  >>> plot(x1, y1, 'g^', x2, y2, 'g-')
  ```

  In this case, any additional keyword argument applies to all
  datasets. Also this syntax cannot be combined with the *data*
  parameter.

By default, each line is assigned a different style specified by a
'style cycle'. The *fmt* and line property parameters are only
necessary if you want explicit deviations from these defaults.
Alternatively, you can also change the style cycle using
:rc:`axes.prop_cycle`.

Parameters
----
x, y : array-like or scalar
    The horizontal / vertical coordinates of the data points.
    x values are optional and default to ``range(len(y))``.

    Commonly, these parameters are 1D arrays.

    They can also be scalars, or two-dimensional (in that case, the
    columns represent separate data sets).

    These arguments cannot be passed as keywords.

fmt : str, optional
    A format string, e.g. 'ro' for red circles. See the *Notes*
    section for a full description of the format strings.

    Format strings are just an abbreviation for quickly setting
    basic line properties. All of these and more can also be

```
            controlled by keyword arguments.

            This argument cannot be passed as keyword.

        data : indexable object, optional
            An object with labelled data. If given, provide the label names to
            plot in *x* and *y*.

            .. note::
                Technically there's a slight ambiguity in calls where the
                second label is a valid *fmt*. ``plot('n', 'o', data=obj)``
                could be ``plt(x, y)`` or ``plt(y, fmt)``. In such cases,
                the former interpretation is chosen, but a warning is issued.
                You may suppress the warning by adding an empty format string
                ``plot('n', 'o', '', data=obj)``.

        Returns
        -------
        list of `.Line2D`
            A list of lines representing the plotted data.

        Other Parameters
        ----------------
        scalex, scaley : bool, default: True
            These parameters determine if the view limits are adapted to the
            data limits. The values are passed on to `autoscale_view`.

        **kwargs : `.Line2D` properties, optional
            *kwargs* are used to specify properties like a line label (for
            auto legends), linewidth, antialiasing, marker face color.
            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')

            If you make multiple lines with one plot call, the kwargs
            apply to all those lines.

            Here is a list of available `.Line2D` properties:

            Properties:
            agg_filter: a filter function, which takes a (m, n, 3) float array and a dpi val
    ue, and returns a (m, n, 3) array
            alpha: float or None
            animated: bool
            antialiased or aa: bool
            clip_box: `.Bbox`
            clip_on: bool
            clip_path: Patch or (Path, Transform) or None
            color or c: color
            contains: unknown
            dash_capstyle: {'butt', 'round', 'projecting'}
            dash_joinstyle: {'miter', 'round', 'bevel'}
            dashes: sequence of floats (on/off ink in points) or (None, None)
            data: (2, N) array or two 1D arrays
            drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}, d
    efault: 'default'
            figure: `.Figure`
            fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
            gid: str
            in_layout: bool
            label: object
            linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq), ...}
            linewidth or lw: float
            marker: marker style string, `~.path.Path` or `~.markers.MarkerStyle`
```

```
        markeredgecolor or mec: color
        markeredgewidth or mew: float
        markerfacecolor or mfc: color
        markerfacecoloralt or mfcalt: color
        markersize or ms: float
        markevery: None or int or (int, int) or slice or List[int] or float or (float, f
loat) or List[bool]
        path_effects: `.AbstractPathEffect`
        picker: unknown
        pickradius: float
        rasterized: bool or None
        sketch_params: (scale: float, length: float, randomness: float)
        snap: bool or None
        solid_capstyle: {'butt', 'round', 'projecting'}
        solid_joinstyle: {'miter', 'round', 'bevel'}
        transform: `matplotlib.transforms.Transform`
        url: str
        visible: bool
        xdata: 1D array
        ydata: 1D array
        zorder: float

    See Also
    --------
    scatter : XY scatter plot with markers of varying size and/or color (
        sometimes also called bubble chart).

    Notes
    -----
    **Format Strings**

    A format string consists of a part for color, marker and line::

        fmt = '[marker][line][color]'

    Each of them is optional. If not provided, the value from the style
    cycle is used. Exception: If ``line`` is given, but no ``marker``,
    the data will be a line without markers.

    Other combinations such as ``[color][marker][line]`` are also
    supported, but note that their parsing may be ambiguous.

    **Markers**

    =============    ===============================
    character        description
    =============    ===============================
    ``'.'``          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
=============   ===============================

**Line Styles**

=============   ===============================
character       description
=============   ===============================
``'-'``         solid line style
``'--'``        dashed line style
``'-.'``        dash-dot line style
``':'``         dotted line style
=============   ===============================

Example format strings::

    'b'    # blue markers with default shape
    'or'   # red circles
    '-g'   # green solid line
    '--'   # dashed line with default color
    '^k:'  # black triangle_up markers connected by a dotted line

**Colors**

The supported color abbreviations are the single letter codes

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

and the ``'CN'`` colors that index into the default property cycle.

If the color is the only part of the format string, you can
additionally use any  `matplotlib.colors` spec, e.g. full names
(``'green'``) or hex strings (``'#008000'``).
```

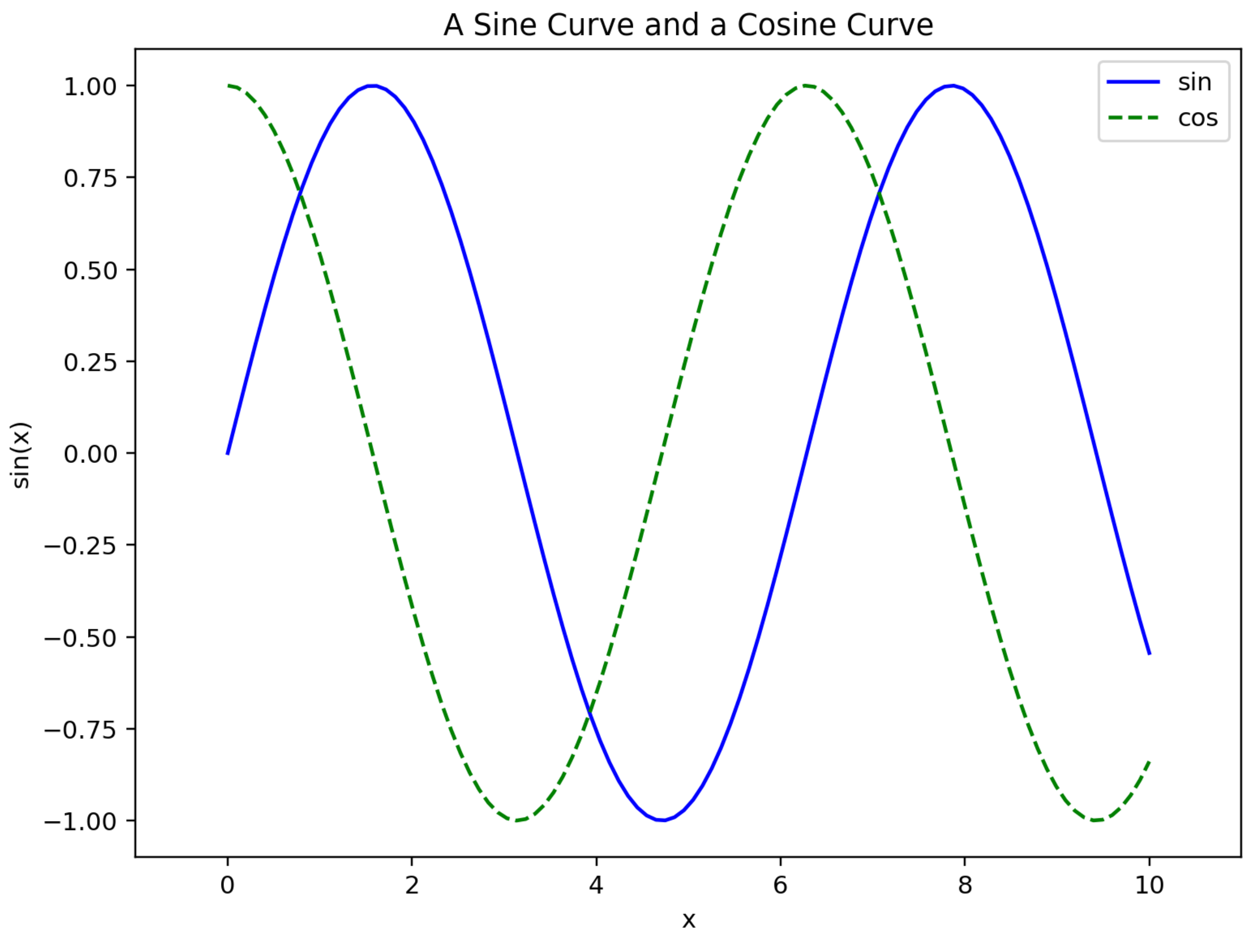
Basic usage of pyplot: Very similiar to Matlab

In [9]:

```
import numpy as np
x = np.linspace(0, 10, 100) #100 evenly spaced points from 0 to 10
fig = plt.figure(figsize=(8, 6),dpi=220) # create the figure, just like figure() in mat
plt.plot(x, np.sin(x), linestyle = '-',color = 'b',label='sin')  # label is used for le
plt.plot(x, np.cos(x), '--g', label = 'cos')
plt.xlim(-1, 11)
plt.title("A Sine Curve and a Cosine Curve")
plt.xlabel('x')
plt.ylabel("sin(x)")
plt.legend()
```

Out[9]:

```
<matplotlib.legend.Legend at 0x253f4785f40>
```



Of course there is some object-oriented feature.

In [10]:
```
type(fig)
```

Out[10]:
```
matplotlib.figure.Figure
```

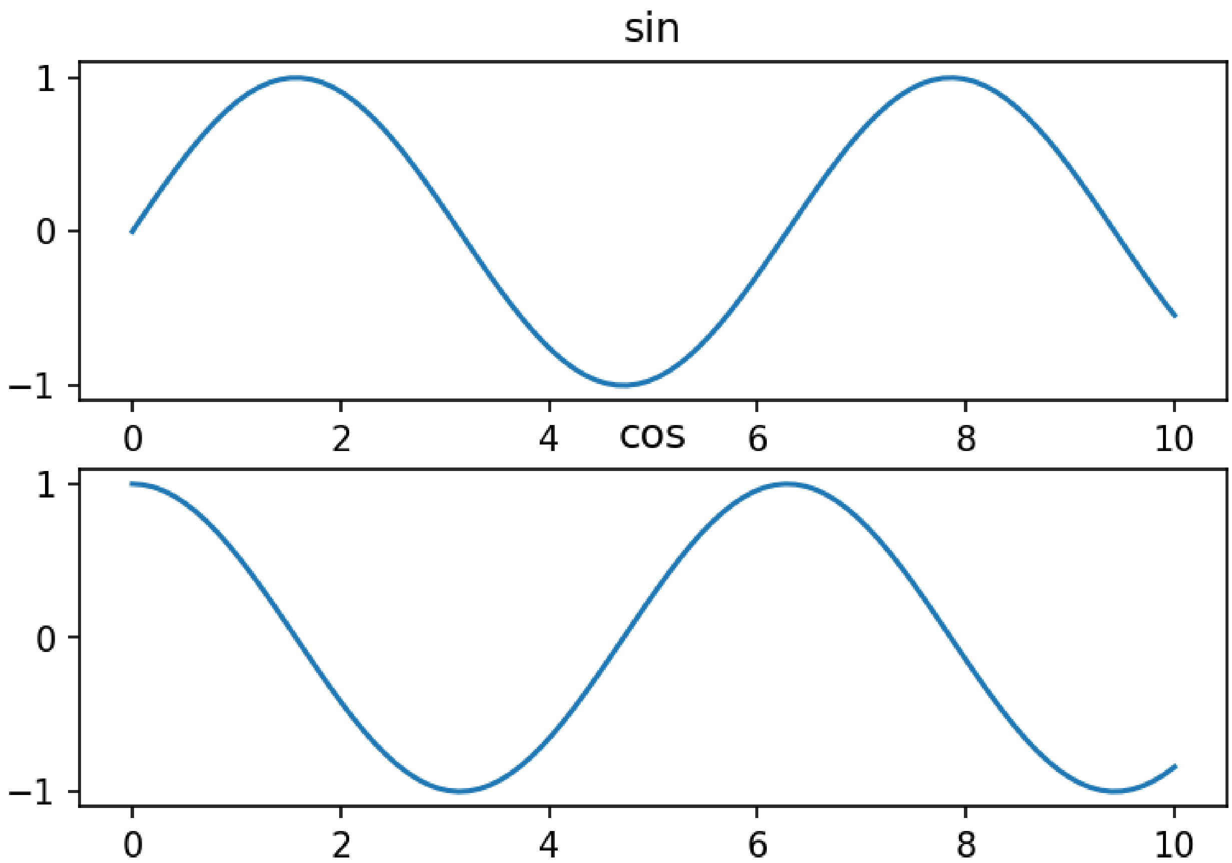
In [ ]:
```
dir(fig)
```

In [12]:
```
fig.savefig('myfigure.png') # savefig is just a method of instance fig!
```

The object-oriented feature is more evident in making subplots. Explore more usages here.

In [16]:
```
# subplots
fig, ax = plt.subplots(2, dpi =150) #dpi is dots per inch, measurement of image resolut
ax[0].plot(x, np.sin(x)) # plot and set_title are the methods of ax[0] -axes
ax[0].set_title('sin')
ax[1].plot(x, np.cos(x))
ax[1].set_title('cos')
```

Out[16]:
```
Text(0.5, 1.0, 'cos')
```



Distinguish the concept of axes and axis in Matplotlib

In [17]:
```
type(ax)
```

Out[17]:
```
numpy.ndarray
```

In [11]:
```
type(ax[0])
```

Out[11]:
```
matplotlib.axes._subplots.AxesSubplot
```

In [18]:
```
fig
```

Out[18]:

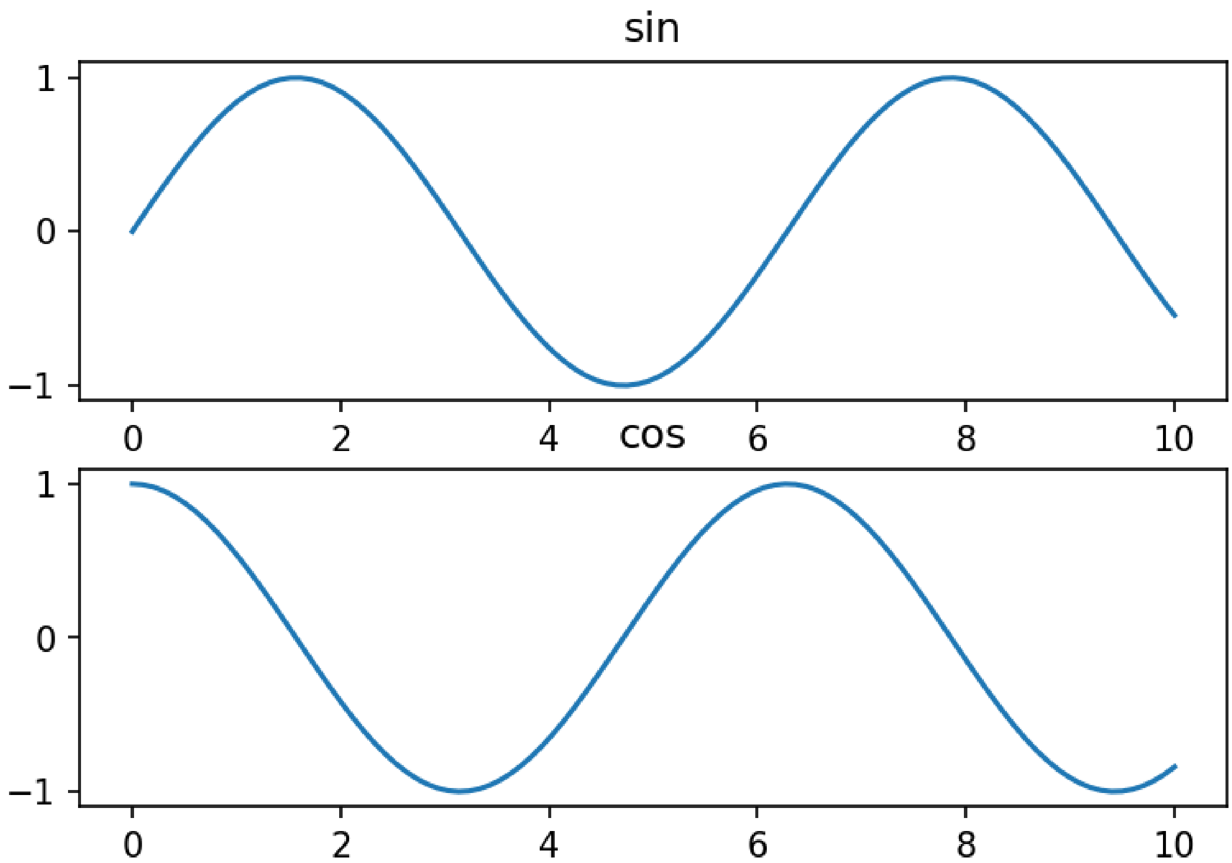


Image Processing

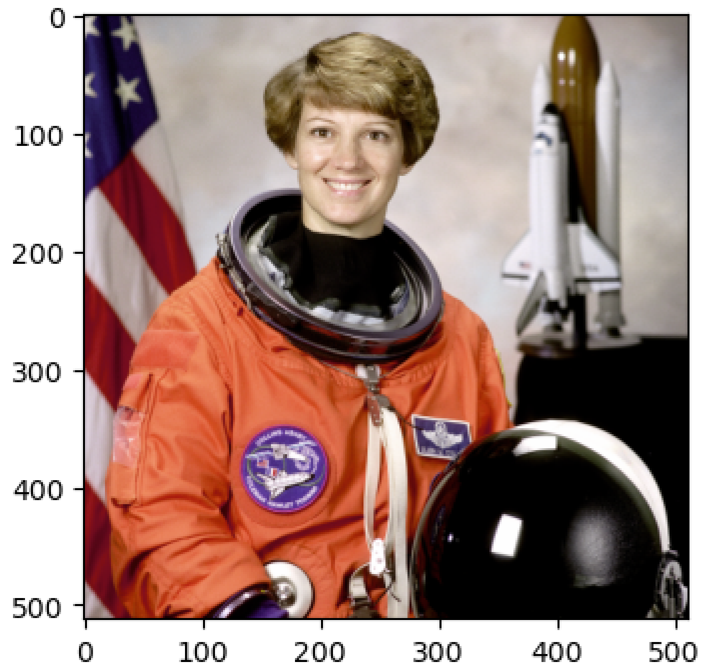
There are many great packages available to handle the image data in Python, such as Pillow, Scikit-Image and opencv-python.

Here we import images from Scikit-Image which is well-compatible with Numpy, and use Numpy to manipulate images.

In [21]:

```
from skimage import data
image_astro = data.astronaut()# read the image as numpy array
image_rock = data.rocket()
fig = plt.figure(dpi=100)
plt.imshow(image_astro)
```

Out[21]: <matplotlib.image.AxesImage at 0x253f6854220>



In [22]:

```
fig = plt.figure(dpi=100)
plt.imshow(image_rock)
```

Out[22]: <matplotlib.image.AxesImage at 0x253f68a7670>

In data science, a common way to store image is through 2D matrix (gray) or 3D tensor (RGB color).

For instance, a gray-scale image with size $m \times n$ can be represented by a matrix $I_1 \in \mathbb{R}^{m \times n}$, whose elements denotes the intensities of pixels.

A color image $m \times n$ can be represented by a tensor (or you can imagine three matrices stacked together) $I_2 \in \mathbb{R}^{m \times n \times 3}$, where the three $m \times n$ matrices denote the intensity in red, green and blue channels respectively (basic assumption is any color can be decomposed in RGB)

In [23]:
```
image_astro.shape # 512-by-512 pixels, with RGB color channels
```

Out[23]:
```
(512, 512, 3)
```

In [24]:
```
image_rock.shape
```

Out[24]:
```
(427, 640, 3)
```

In [25]:
```
image_rock[0,0,] # the RGB of first pixel
```

Out[25]:
```
array([17, 33, 58], dtype=uint8)
```

In [26]:
```
[np.max(image_astro),np.min(image_astro)]
```

Out[26]:
```
[255, 0]
```

Even with simple Numpy expressions, you can do some image processing like in Photoshop!

- Crop the images

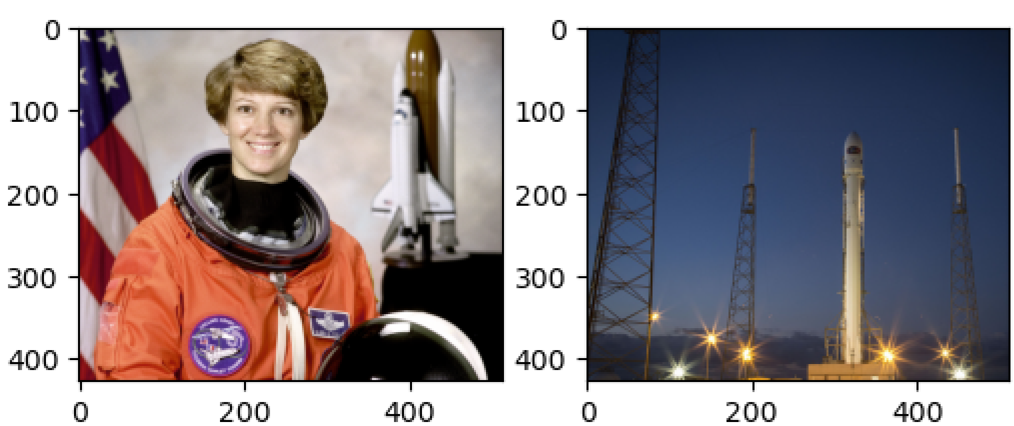
In [27]:
```
image_astro_split = image_astro[:427,:,:]
image_rock_split = image_rock[:,:512,:]
```

In [29]:
```
image_rock_split.shape
```

Out[29]:
```
(427, 512, 3)
```

In [30]:
```
fig, ax = plt.subplots(ncols=2, dpi = 100)
ax[0].imshow(image_astro_split) # plot and set_title are the methods of ax[0] -axes
ax[1].imshow(image_rock_split)
```

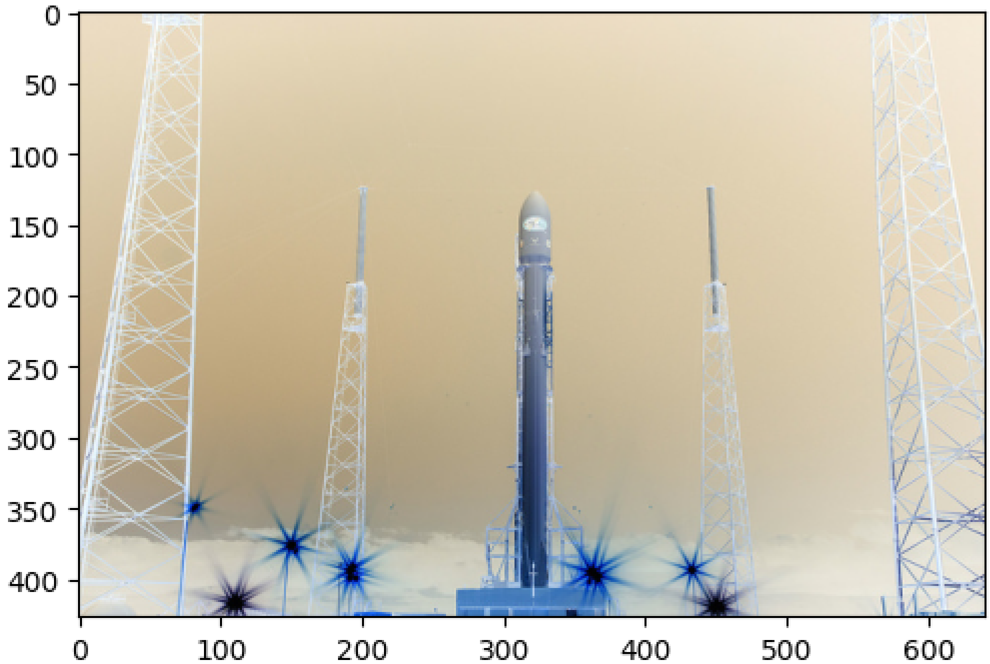
Out[30]:
```
<matplotlib.image.AxesImage at 0x253f690f7f0>
```



- Invert the color intensities

In [23]:
```
fig = plt.figure(dpi=100)
plt.imshow(255-image_rock)
```

Out[23]: <matplotlib.image.AxesImage at 0x7fcf290934d0>



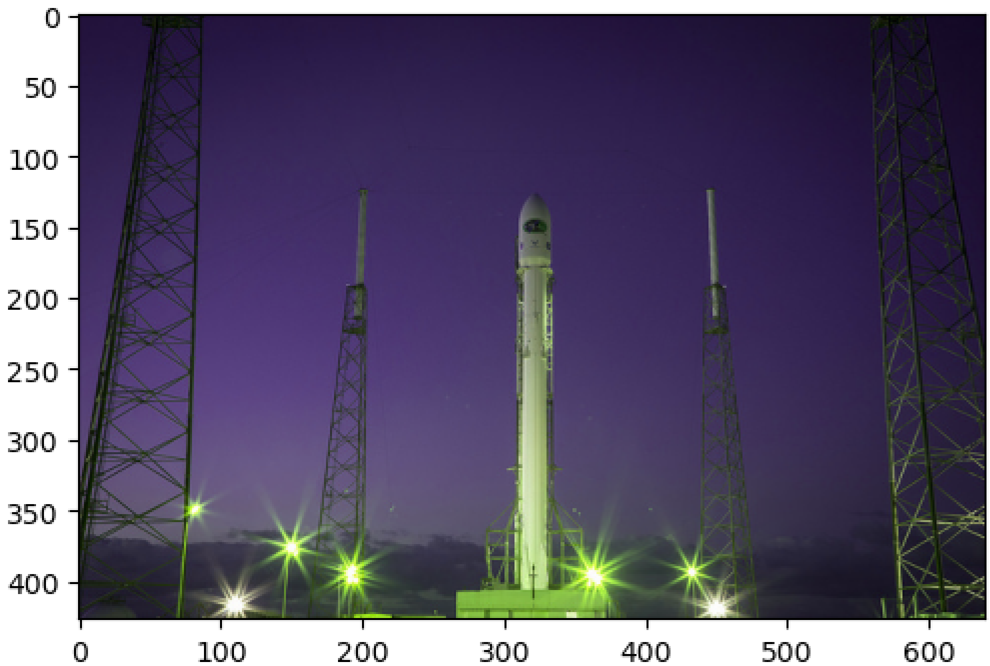
- Exchange RGB channels

In [43]:
```
fig = plt.figure(dpi=100)
plt.imshow(image_rock[:,:,[0,2,1]])
```

Out[43]: <matplotlib.image.AxesImage at 0x253f7fb76a0>

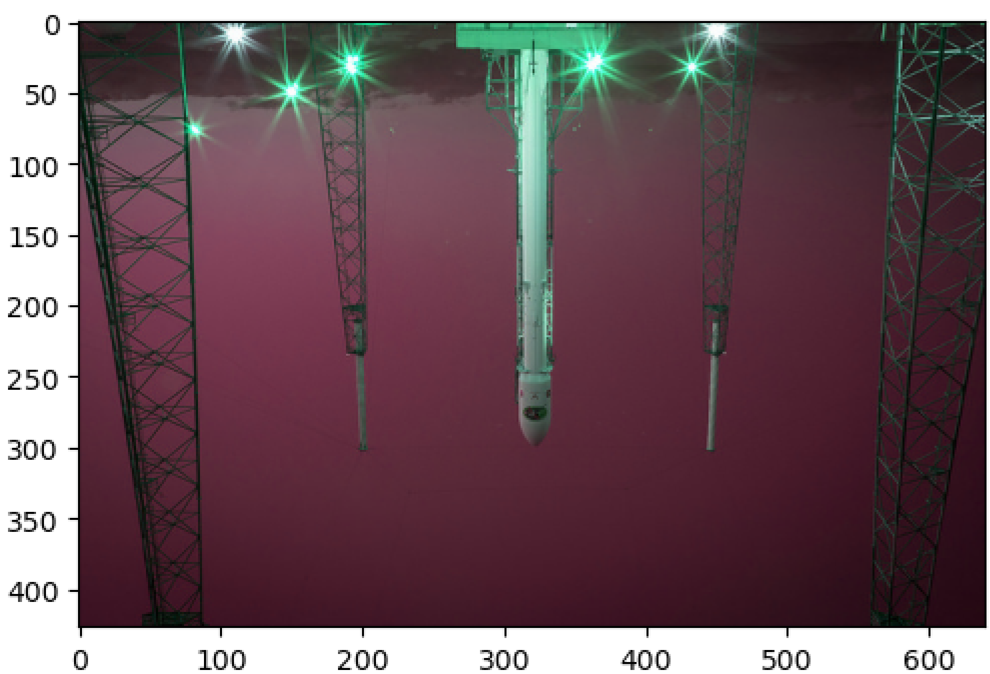
In [46]:
```
fig = plt.figure(dpi=100)
plt.imshow(image_rock[:,:,[1,0,2]])
```

Out[46]: <matplotlib.image.AxesImage at 0x253f80d6af0>



In [48]:
```
fig = plt.figure(dpi=100)
plt.imshow(image_rock[::-1,:,[2,0,1]])
```

Out[48]: <matplotlib.image.AxesImage at 0x253f6bb67f0>



- Binarize the image

In [37]:
```
image = image_rock
```

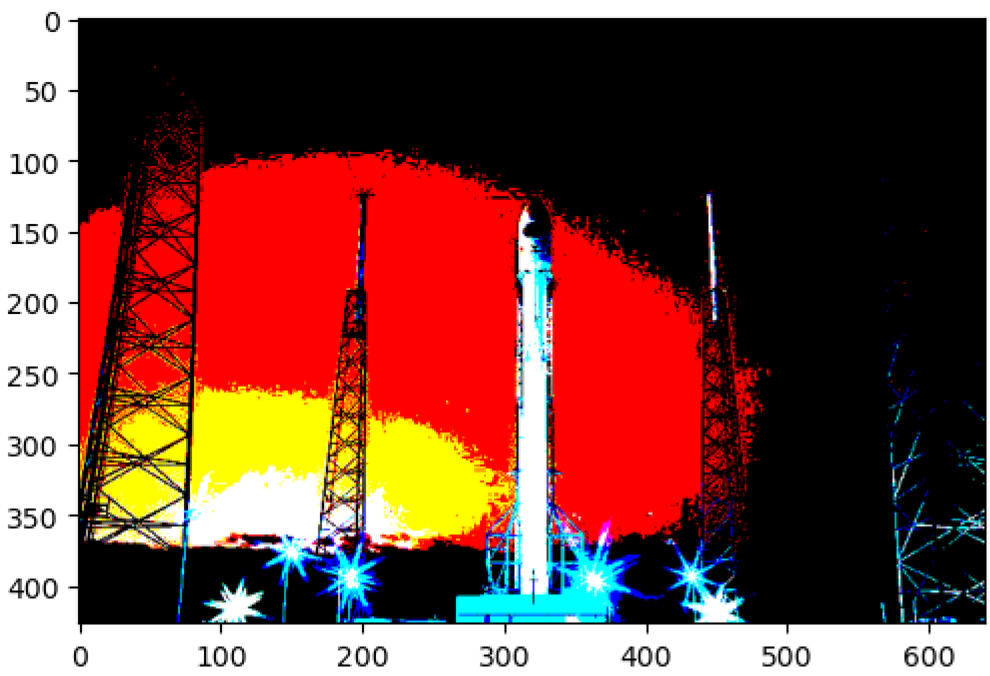
```python
image_bi = np.empty_like(image)

thresh = 90
maxval = 255

for i in range(3): #loop over each color channel
    image_bi[:, :, i] = (image[:, :, i] > thresh) * maxval

fig = plt.figure(dpi=100)
plt.imshow(image_bi[:,:,[2,1,0]])
```

Out[37]: <matplotlib.image.AxesImage at 0x253f6bd7eb0>



In [31]:

```python
image_bi
```

Out[31]:

```
array([[[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],

       [[  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0],
        ...,
        [  0,   0,   0],
        [  0,   0,   0],
        [  0,   0,   0]],
```

```
        ...,

        [[  0,   0,   0],
         [  0,   0,   0],
         [  0,   0,   0],
         ...,
         [255, 255,   0],
         [255, 255,   0],
         [255, 255,   0]],

        [[  0,   0,   0],
         [  0,   0,   0],
         [  0,   0,   0],
         ...,
         [255,   0,   0],
         [255,   0,   0],
         [  0,   0,   0]],

        [[  0,   0,   0],
         [  0,   0,   0],
         [  0,   0,   0],
         ...,
         [  0,   0,   0],
         [  0,   0,   0],
         [  0,   0,   0]]], dtype=uint8)
```

- Blending

In [42]:

```
image_combine = 0.2*image_astro_split+0.5*image_rock_split
fig = plt.figure(dpi=100)
plt.imshow(image_combine.astype('uint8'))
plt.axis('off')
```

Out[42]: (-0.5, 511.5, 426.5, -0.5)

In [ ]: