EE 538 Autumn 2020 Week 1

We'll use **numpy** in nearly everything we do, so let's use the industry standard **np** when importing. This allows us to call numpy's modules using this shorthand prefix.

In [55]:

```
import numpy as np
```

The **scipy** module **signal** contains many useful signal processing functions:

In [56]:

```
from scipy import signal
```

For plotting/visualization, **matplotlib** is indispensable. We'll use the shorthand **plt** for quickly creating plots:

In [57]:

```
import matplotlib
import matplotlib.pyplot as plt
```

pandas offers many powerful tools for file I/O and data manipulation. We may not use it much here, but it's definitely a package to become familiar with (it's used ubiquitously in data science, for example).

In [58]:

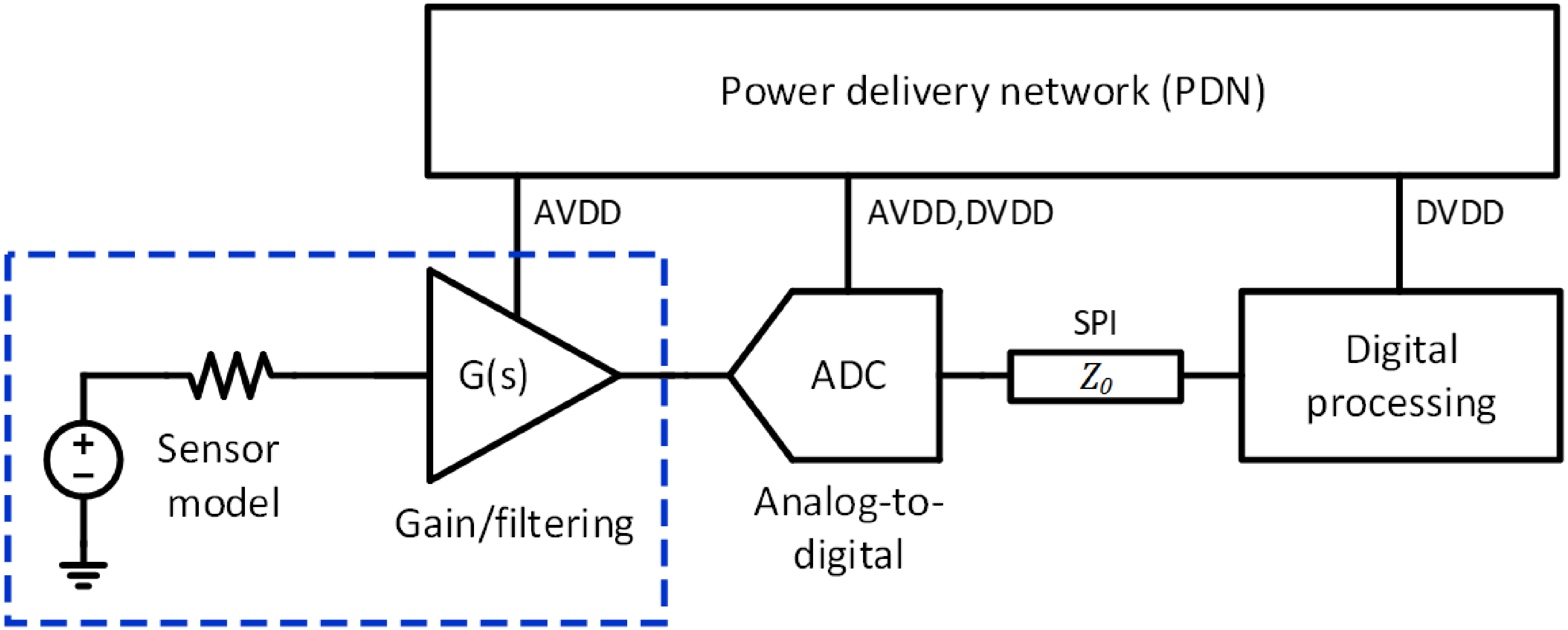
```
import pandas as pd
```

We'll perform certain functions (like plotting signals) frequently, so it makes sense to encapsulate these in reusable modules:

In [59]:

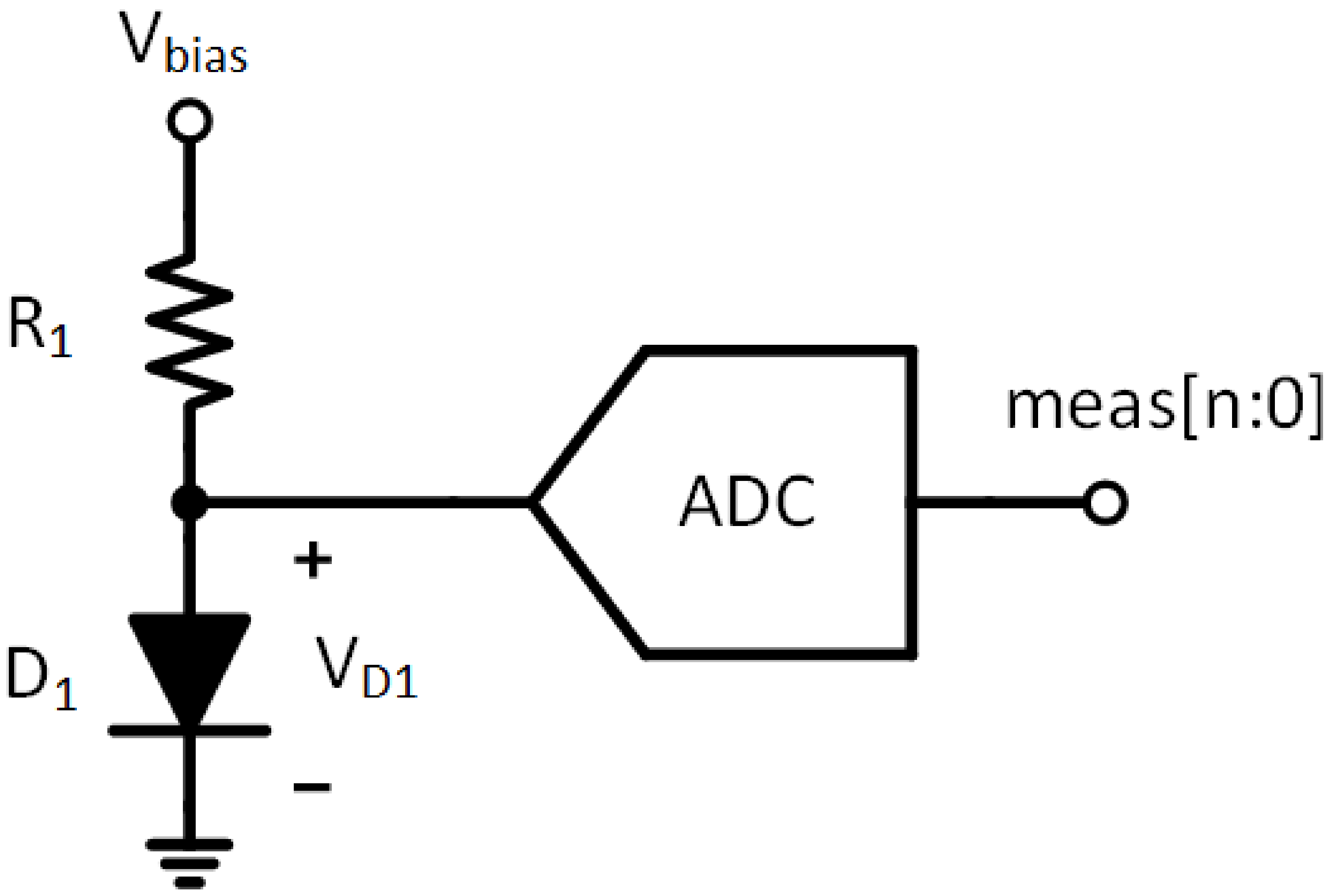
```
def plot_xy(x, y, xlabel, ylabel):
    fig, ax = plt.subplots(figsize = (10.0, 7.5))
    ax.plot(x, y)
    ax.grid()
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
```

Sensor Systems



Signals

Temperature sensor using a pn-junction



- The voltage across a pn junction (diode) is given by

$$V_{D1} = \frac{kT}{q} \ln \frac{I_D}{I_S}$$

- Thus, if I_D is constant, V_{D1} has a temperature slope of

$$\frac{dV}{dT} = \frac{k}{q} \ln \frac{I_D}{I_S}$$

- Let's plot this...

In [60]:

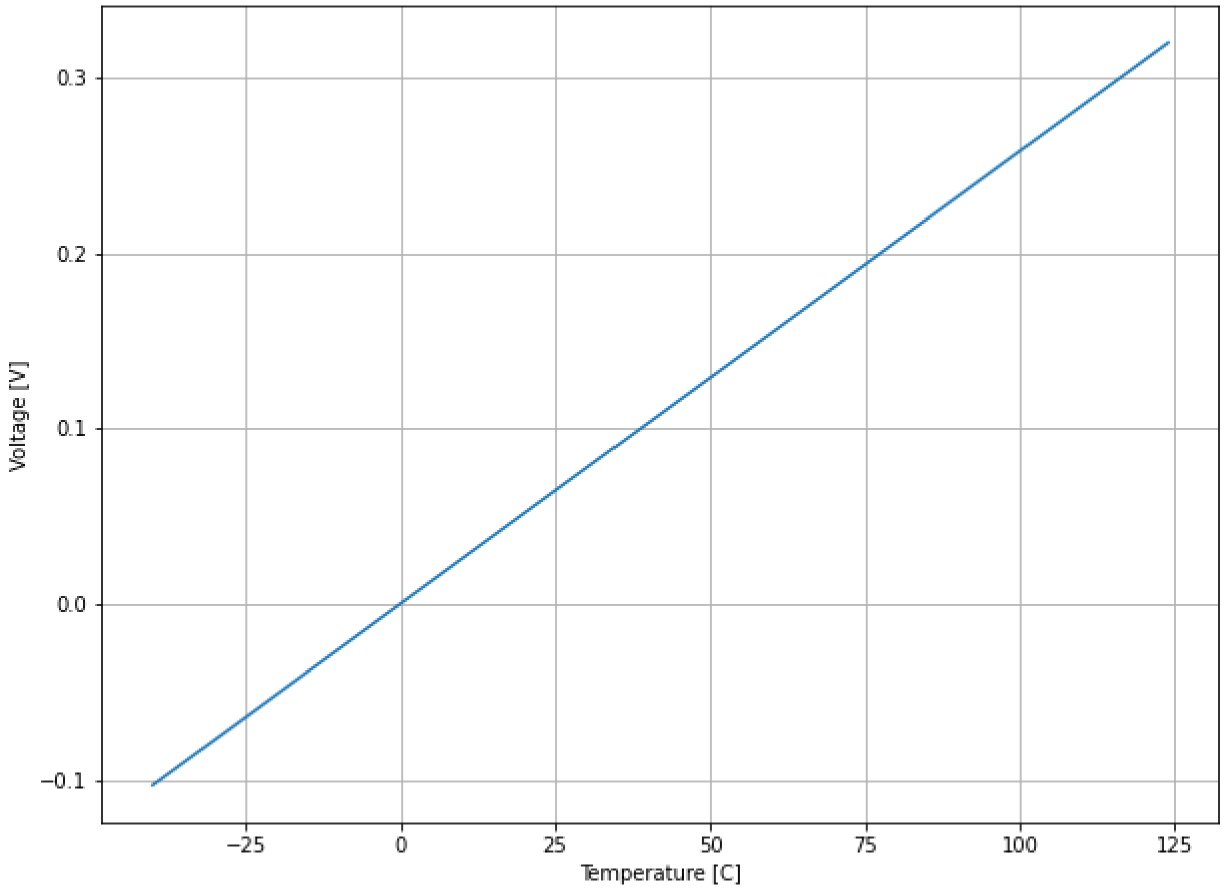
```python
# physical constants
Is = 1e-16
k = 1.38e-23
q = 1.602e-19

# assume a constant diode current of 1mA
Id = 1e-3

# Temperature slope of diode voltage
dV_dT = k*np.log(Id/Is)/q

# define temperature range
T = np.arange(-40, 125, step=1)
```

In [61]:
```
# plot diode voltage versus temperature
plot_xy(T, dV_dT*T, 'Temperature [C]', 'Voltage [V]')
```



Plot V-I relationship for a 1-kΩ resistor

$$V = IR$$

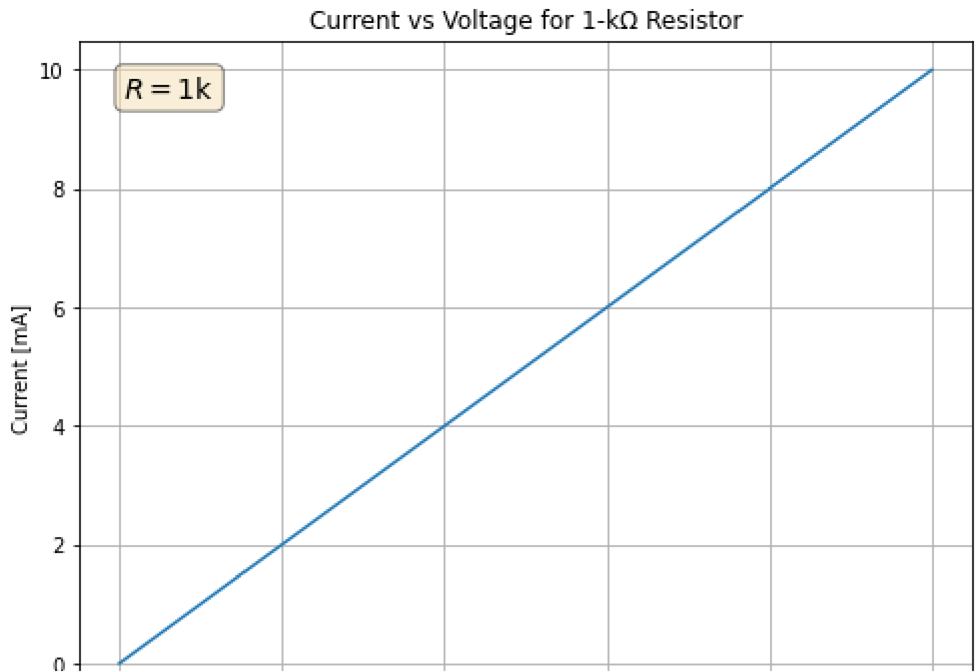
In [62]:

```
R = 1e3
C = 1e-6
V = np.linspace(0, 10, num=100, endpoint=True) # linear sweep of voltage
I = V/R

fig, ax = plt.subplots(figsize = (8.0, 6.0))
ax.plot(V, 1e3*I)
ax.set_ylabel('Current [mA]')
ax.set_xlabel('Voltage [V]')
ax.set_title('Current vs Voltage for 1-k\u03A9 Resistor')
ax.grid()
textstr = r'$R=%.0f$k' % (R/1e3, )
props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize =14,
            verticalalignment='top', bbox=props)
```

Out[62]:

```
Text(0.05, 0.95, '$R=1$k')
```



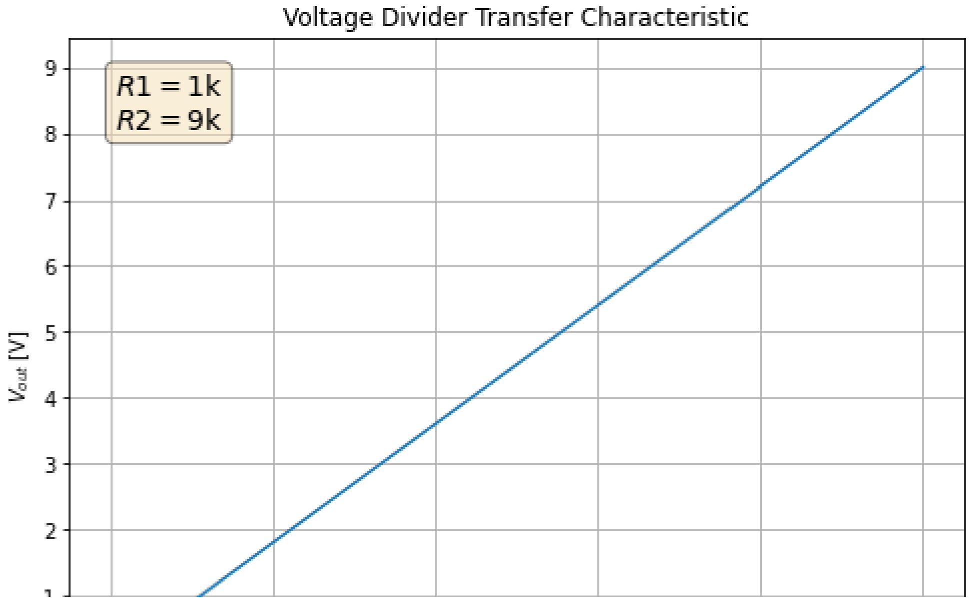

0
2
4
6
8
10
Voltage [V]

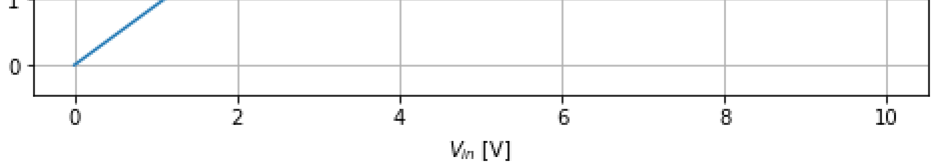
Voltage divider

In [63]:

```python
R1 = 1e3
R2 = 9e3
Vin = np.linspace(0, 10, num=100, endpoint=True) # linear sweep of voltage
Vout = Vin*R2/(R1+R2)

fig, ax = plt.subplots(figsize = (8.0,6.0))
ax.plot(Vin, Vout)
ax.set_ylabel('$V_{out}$ [V]')
ax.set_yticks(np.arange(0, 10, step=1))
ax.set_xlabel('$V_{in}$ [V]')
ax.set_title('Voltage Divider Transfer Characteristic')
ax.grid()
textstr = '\n'.join((
    r'$R1=%.0f$k' % (R1/1e3, ),
    r'$R2=%.0f$k' % (R2/1e3, )))
props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize =14,
            verticalalignment='top', bbox=props)
plt.show()
```



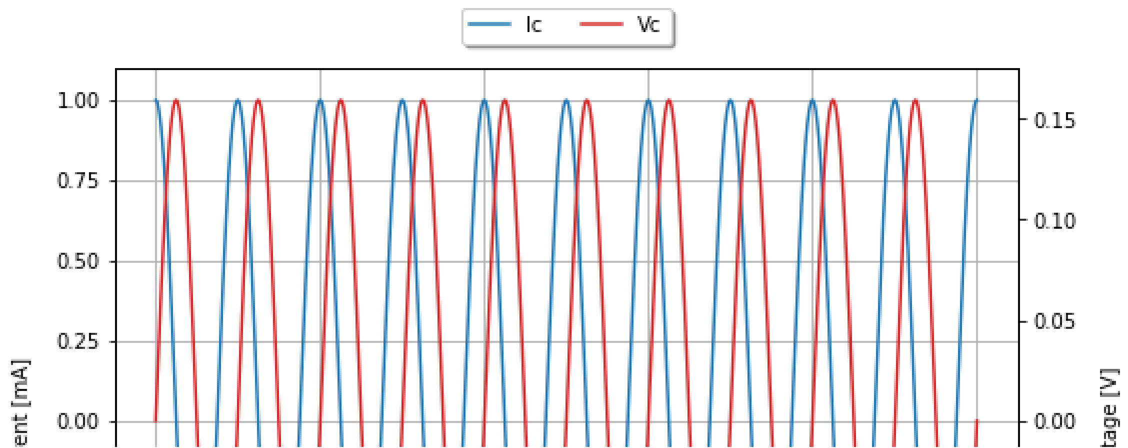

1
0
0
2
4
6
8
10
V_{in} [V]

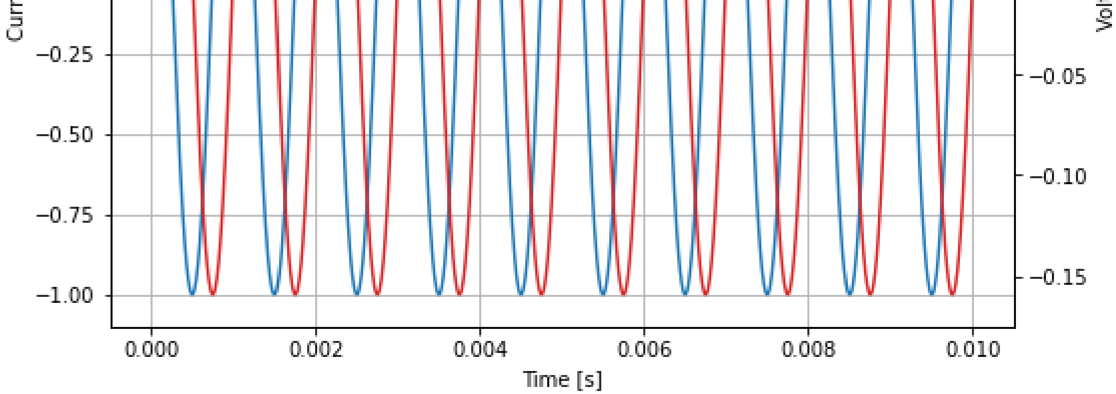
Capacitor voltage-current relationship

In [64]:

```
# Create a sinusoidal current and simulate capacitor voltage
t = np.linspace(0, 10e-3, num=1000, endpoint=True)
f = 1e3
w = 2*np.pi*f
isin = 1e-3*np.cos(w*t)
Yc = signal.TransferFunction([1], [1E-6, 0])
tout, vsin, x = signal.lsim(Yc, isin, t, X0=0, interp=1)

# Plot the input and output with separate y axes
fig, ax1 = plt.subplots(figsize = (8.0, 6.0))
lns1 = ax1.plot(t, 1e3*isin, color='tab:blue', label='Ic')
ax2 = ax1.twinx()
lns2 = ax2.plot(tout, vsin, color='tab:red', label='Vc')
lns = lns1 + lns2
labs = [l.get_label() for l in lns]
ax1.legend(lns, labs, loc='upper center', ncol=2, fancybox=True,
           shadow=True, bbox_to_anchor=(0.5,1.1) )
ax1.set_ylabel('Current [mA]')
ax1.set_xlabel('Time [s]')
ax1.grid()
ax2.set_ylabel('Voltage [V]')
plt.tight_layout()
plt.show()
```





Curr
−0.25
−0.50
−0.75
−1.00
0.000
0.002
0.004
0.006
0.008
0.010
Time [s]
Vol
−0.05
−0.10
−0.15

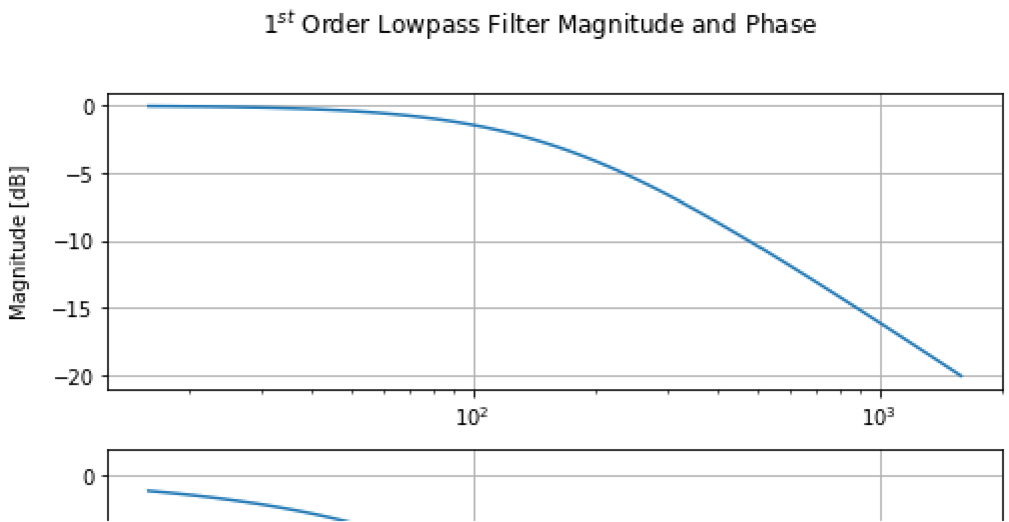
Simple RC Filter

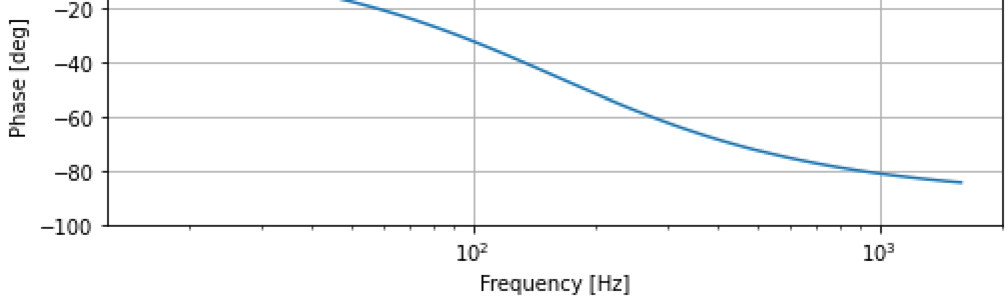
In [65]:

```
R = 1e3
C = 1e-6
tau = R*C

""" Lowpass filter frequency response"""
filt_lp = signal.TransferFunction([1], [tau, 1])
w, mag, phase = filt_lp.bode()        # rad/s, dB, degrees
f = w/2/np.pi

# Plot the frequency response
fig, axs = plt.subplots(2, figsize = (8.0, 6.0))
fig.suptitle('$1^{st}$ Order Lowpass Filter Magnitude and Phase')
axs[0].semilogx(f, mag)
axs[0].grid()
axs[0].set_ylabel('Magnitude [dB]')
axs[1].semilogx(f,phase)
axs[1].grid()
axs[1].set_ylabel('Phase [deg]')
axs[1].set_xlabel('Frequency [Hz]')
axs[1].set_ylim(-100, 10)
fig.align_ylabels(axs[:])
```



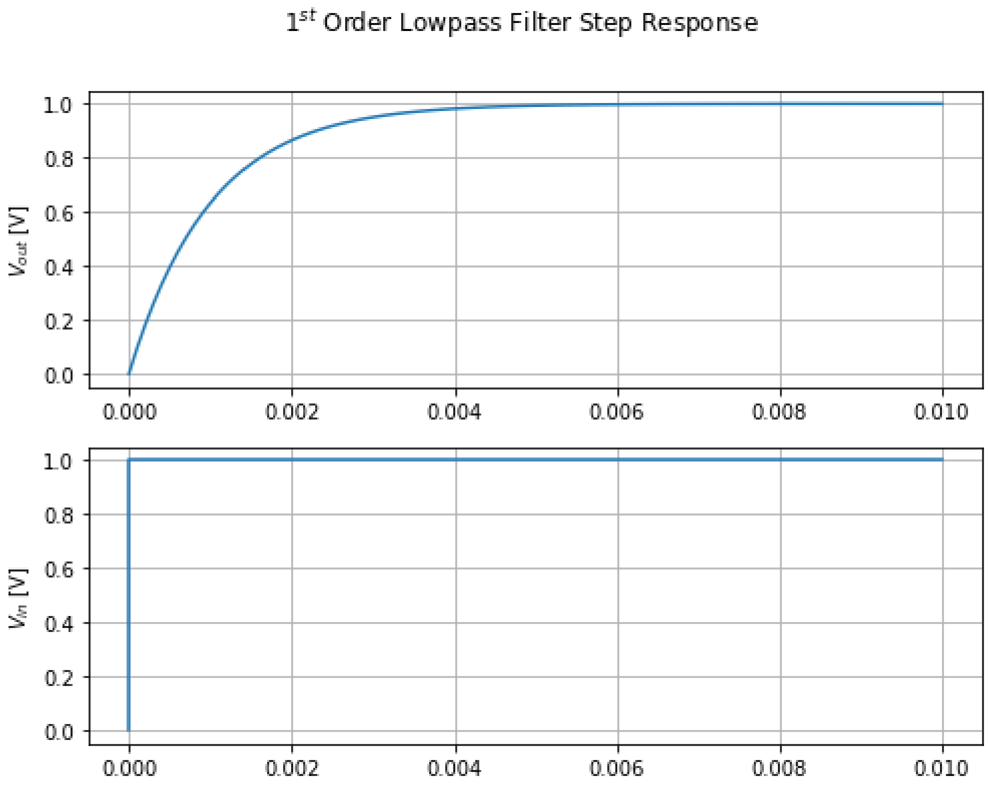


Phase [deg]
−20
−40
−60
−80
−100
10^2
10^3
Frequency [Hz]

Lowpass filter step response

In [66]:
```
tin = np.linspace(0,10e-3,100)
u_step = np.concatenate( (0, np.ones(99)), axis=None)
tout,vout = signal.step(filt_lp, X0=None, T=tin)

# Plot the simulation result
fig, axs = plt.subplots(2, figsize=(8.0, 6.0))
fig.suptitle('$1^{st}$ Order Lowpass Filter Step Response')
axs[0].plot(tout, vout)
axs[0].set_ylabel('$V_{out}$ [V]')
axs[0].grid()
axs[1].step(tin, u_step)
axs[1].set_ylabel('$V_{in}$ [V]')
axs[1].set_xlabel('Time [s]')
axs[1].grid()
```

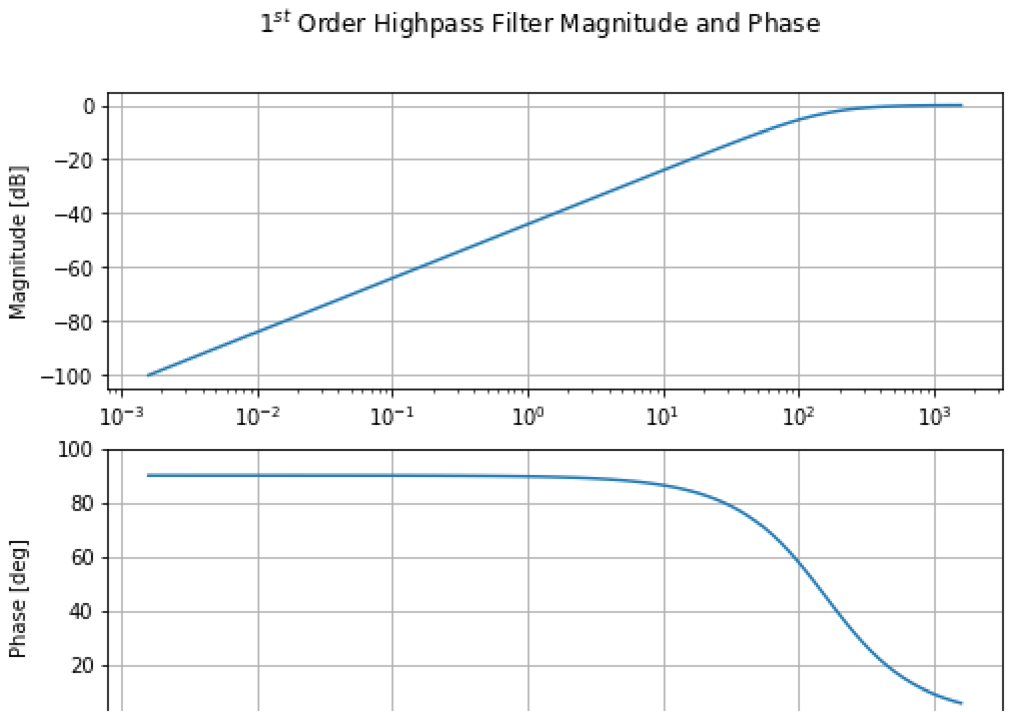


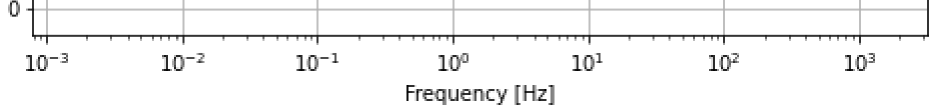
Time [s]

Highpass filter frequency response

In [67]:
```
filt_hp = signal.TransferFunction([tau, 0], [tau, 1])
w, mag, phase = filt_hp.bode()      # rad/s, dB, degrees
f = w/2/np.pi

# Plot the frequency response
fig, axs = plt.subplots(2, figsize = (8.0, 6.0))
fig.suptitle('$1^{st}$ Order Highpass Filter Magnitude and Phase')
axs[0].semilogx(f, mag)
axs[0].grid()
axs[0].set_ylabel('Magnitude [dB]')
axs[1].semilogx(f,phase)
axs[1].grid()
axs[1].set_ylabel('Phase [deg]')
axs[1].set_xlabel('Frequency [Hz]')
axs[1].set_ylim(-10, 100)
fig.align_ylabels(axs[:])
```



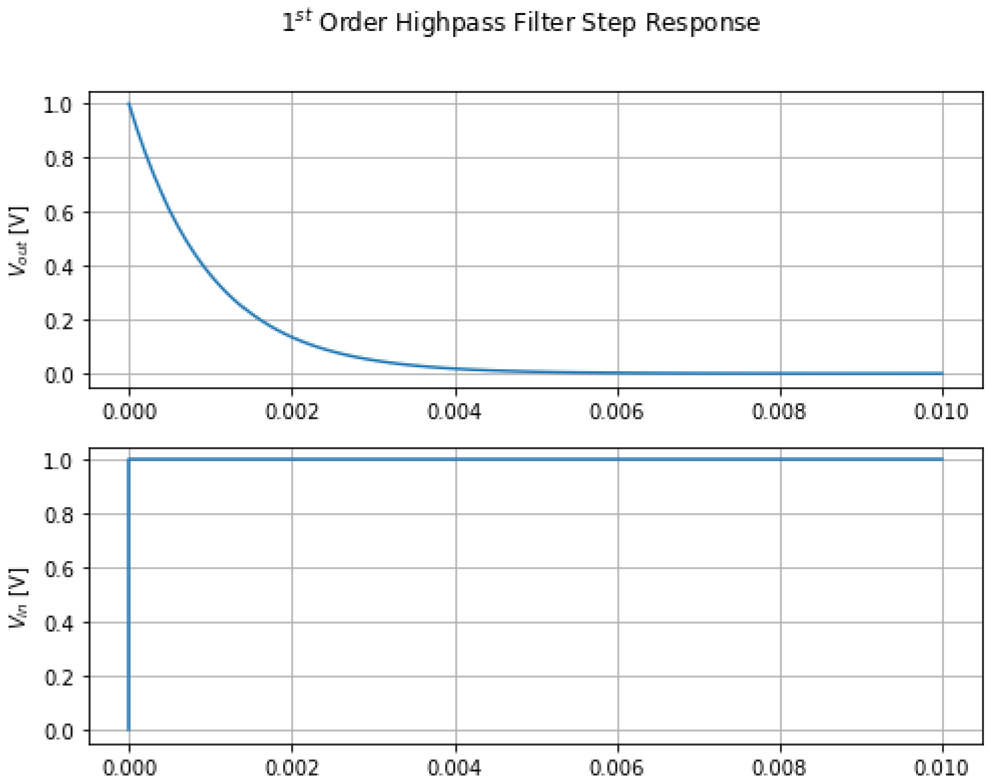

0
10⁻³
10⁻²
10⁻¹
10⁰
10¹
10²
10³
Frequency [Hz]

Highpass filter step response

In [68]:

```
tin = np.linspace(0,10e-3,100)
u_step = np.concatenate( (0, np.ones(99)), axis=None)
tout,vout = signal.step(filt_hp, X0=None, T=tin)

# Plot the simulation result
fig, axs = plt.subplots(2, figsize=(8.0, 6.0))
fig.suptitle('$1^{st}$ Order Highpass Filter Step Response')
axs[0].plot(tout, vout)
axs[0].set_ylabel('$V_{out}$ [V]')
axs[0].grid()
axs[1].step(tin, u_step)
axs[1].set_ylabel('$V_{in}$ [V]')
axs[1].set_xlabel('Time [s]')
axs[1].grid()
```



Time [s]

FFT of input/output for a LP filter

In [69]:

```
f = 1e3
w = 2*np.pi*f
t_sim = 10e-3                 # length of simulation in seconds
n_sim = 1000                  # number of simulation time points
t_in = np.linspace(0, t_sim, n_sim, endpoint=True) # array of time points
v_in = np.sin(w*t_in)         # sinusoidal "voltage"

# Response of LP filter to sinusoidal input
t_out, v_out, x = signal.lsim(filt_lp, v_in, t_in, X0=None, interp=None)
```

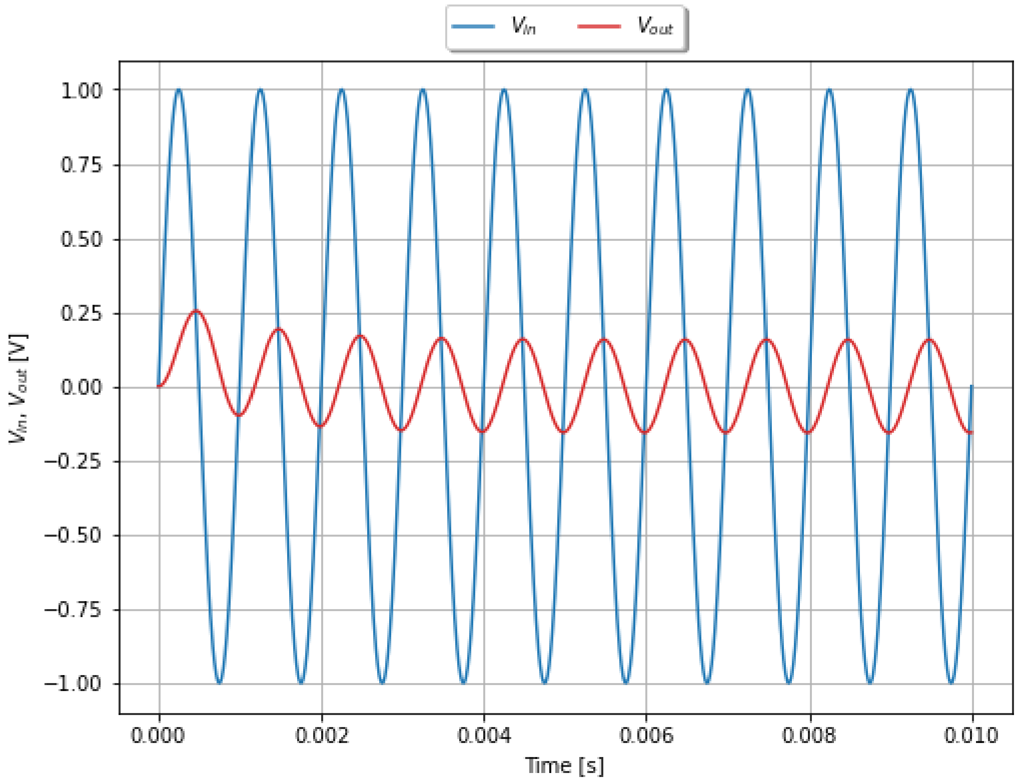
Time domain

In [70]:

```python
# Plot the input and output in the time domain
fig, ax = plt.subplots(figsize = (8.0,6.0))
lns1 = ax.plot(t_in, v_in, color='tab:blue', label='$V_{in}$')
lns2 = ax.plot(t_out, v_out, color='tab:red', label='$V_{out}$')
ax.grid()
lns = lns1 + lns2
labs = [l.get_label() for l in lns]
ax.legend(lns, labs, loc='upper center', ncol=2, fancybox=True,
          shadow=True, bbox_to_anchor=(0.5,1.1) )
ax.set_ylabel('$V_{in}$, $V_{out}$ [V]')
ax.set_xlabel('Time [s]')
```

Out[70]:

```
Text(0.5, 0, 'Time [s]')
```

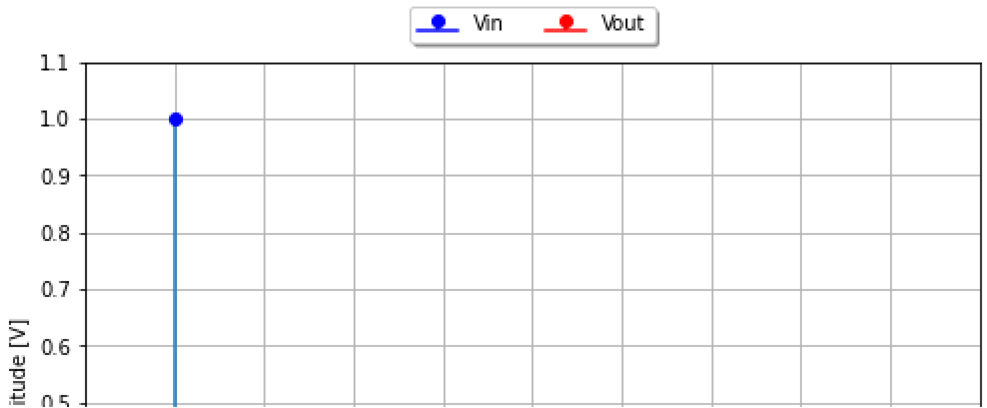


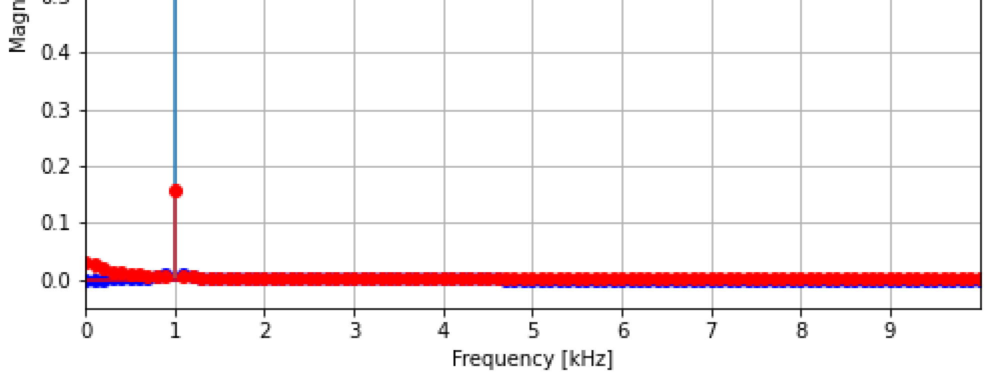
Frequency domain

In [71]:
```
dt = t_sim/n_sim  # time step
fft_v_in = np.fft.rfft(v_in) * 2 / len(v_in) # single-sided FFT
fft_mag_in = np.abs(fft_v_in)                 # FFT magnitude
fft_freq = np.fft.rfftfreq(len(v_out), dt)

fft_v_out = np.fft.rfft(v_out) * 2 / len(v_out) # single-sided FFT
fft_mag_out = np.abs(fft_v_out)                 # FFT magnitude

# Plot the FFT magnitudes using stem
fig, ax = plt.subplots(figsize = (8.0,6.0))
stem1 = ax.stem(1e-3*fft_freq, fft_mag_in, linefmt='tab:blue', markerfmt='bo',
                basefmt='b', label='$V_{in}$', use_line_collection=True)
stem2 = ax.stem(1e-3*fft_freq, fft_mag_out, linefmt='tab:red', markerfmt='ro',
                basefmt='r', label='$V_{out}$', use_line_collection=True)
stem1.set_label('Vin')
stem2.set_label('Vout')
ax.legend(loc='upper center', ncol=2, fancybox=True,
           shadow=True, bbox_to_anchor=(0.5,1.1))
ax.set_xlim(0, 10)
ax.set_xticks(np.arange(0, 10, step=1))
ax.set_yticks(np.arange(0, 1.2, step=.1))
ax.set_xlabel('Frequency [kHz]')
ax.set_ylabel('Magnitude [V]')
ax.grid()
```





Magn
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
5
6
7
8
9
Frequency [kHz]

In [ ]:

```

```

In [ ]:

```

```