(Custom CSS files are not reliable for controlling Jupyter font style. To establish the same appearance as the original notebook, depend on the browser to control the font, by setting the desired font faces in the browser settings. For example, Chrome 135 or Firefox 134 can do this. In this notebook series, Bookerly font is for markdown and Monaco is for code.)

Chapter 1: Basic Concepts
Learning some basic things about Ordinary Differential Equations, and learning how to do some elementary calculations with their factors.

1.3. Determine whether $y(x) = 2e^{-x} + xe^{-x}$ is a solution of $y'' + 2y' + y = 0$

First it is necessary to solve the problem in Python.

In [14]:
```
import sympy as sp
from sympy import *
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']
```

In [15]:
```
x = symbols("x")
y = Function("y")(x)
y
```

Out[15]: $y(x)$

In [16]:
```
y.diff()
```

Out[16]: $\frac{d}{dx} y(x)$

In [16]:
```
ode = Eq(y.diff(x,x) + 2*y.diff(x) + y , 0)
ode
```

Out[16]: $y(x) + 2\frac{d}{dx} y(x) + \frac{d^2}{dx^2} y(x) = 0$

In [17]:
```
sol = dsolve(ode, y)
sol
```

Out[17]: $y(x) = (C_1 + C_2 x)\, e^{-x}$

Then to check the candidate solution.

In [18]:
```
prob = 2*exp(-x) + x*exp(-x)
prob
```

Out[18]: $xe^{-x} + 2e^{-x}$

In [19]:
```
checkodesol(ode, prob)
```

Out[19]: (True, 0)

The candidate solution is verified.

1.4. Determine whether $y(x) = 1$ is a solution of $y'' + 2y' + y = x$

In [20]:
```
import sympy as sp
from sympy import *
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']
```

In [21]:
```
x = symbols("x")
y = Function("y")(x)
y
```

Out[21]: $y(x)$

In [22]:
```
y.diff()
```

Out[22]: $\frac{d}{dx}y(x)$

In [23]:
```
ode = Eq(y.diff(x,x) + 2*y.diff(x) + y , 0)
ode
```

Out[23]: $y(x) + 2\frac{d}{dx}y(x) + \frac{d^2}{dx^2}y(x) = 0$

In [24]:
```
sol = dsolve(ode, y)
sol
```

Out[24]: $y(x) = (C_1 + C_2 x)\, e^{-x}$

Then to check the candidate solution.

In [25]:
```
prob2 = 1
prob2
```

Out[25]: 1

In [26]:
```
checkodesol(ode, prob2)
```

Out[26]: (False, 1)

The candidate is not a solution of the specified ode.

1.5. Show that $y = \ln x$ is a solution of $xy'' + y' = 0$ on $\mathcal{J} = (0, \infty)$ but is not a solution on $\mathcal{J} = (-\infty, \infty)$.

In [2]:
```
import sympy as sp
from sympy import *
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']
```

In [3]:
```
x = symbols("x")
y = Function("y")(x)
y
```

Out[3]: $y(x)$

In [4]:
```
y.diff()
```

Out[4]: $\frac{d}{dx} y(x)$

In [5]:
```
ode = Eq(x*y.diff(x,x) + y.diff(x) , 0)
ode
```

Out[5]: $x\frac{d^2}{dx^2} y(x) + \frac{d}{dx} y(x) = 0$

In [6]:
```
sol = dsolve(ode, y)
sol
```

Out[6]: $y(x) = C_1 + C_2 \log(x)$

Then to check the candidate solution.

In [7]:
```
prob3 = log(x)
prob3
```

Out[7]: $\log(x)$

In [8]:
```
checkodesol(ode, prob3)
```

Out[8]: (True, 0)

Scipy vouches for the log function as solution. This is not really the answer to the problem, however. Maybe it would be helpful to run the following:

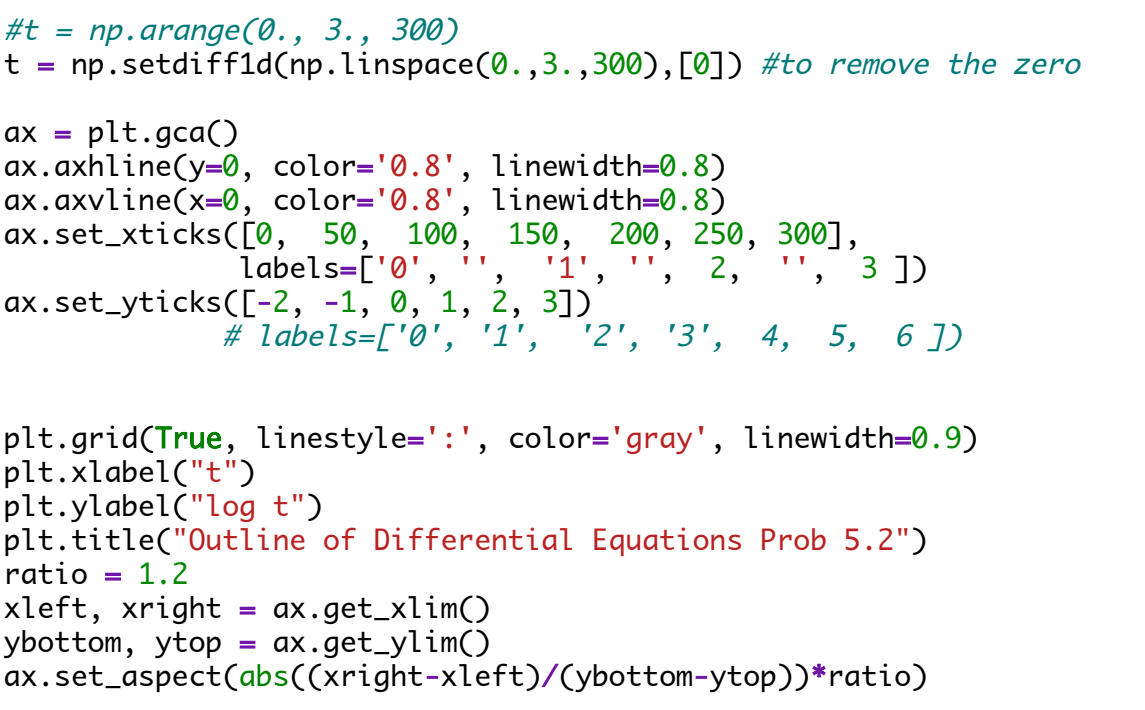
In [9]:
```
solveset(sol, x, Interval(-oo,oo))
```

Out[9]: $\{x \mid x \in (-\infty, \infty) \wedge -C_1 - C_2 \log(x) + y(x) = 0\}$

Since the set of all real x equal to 0 or less is excluded from the described intersection (because of nonexistent ln value), the above effectively restricts the domain according to the rule suggested in the problem. A weasel answer perhaps.

A little trick below for removing 0 from the domain of the plot seems to be working.

In [44]:
```
import numpy as np
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']

#t = np.arange(0., 3., 300)
t = np.setdiff1d(np.linspace(0.,3.,300),[0]) #to remove the zero

ax = plt.gca()
ax.axhline(y=0, color='0.8', linewidth=0.8)
ax.axvline(x=0, color='0.8', linewidth=0.8)
ax.set_xticks([0,  50,  100,  150,  200, 250, 300],
              labels=['0', '',  '1', '',  2,  '',  3 ])
ax.set_yticks([-2, -1, 0, 1, 2, 3])
             # labels=['0', '1',  '2', '3',  4,  5,  6 ])


plt.grid(True, linestyle=':', color='gray', linewidth=0.9)
plt.xlabel("t")
plt.ylabel("log t")
plt.title("Outline of Differential Equations Prob 5.2")
ratio = 1.2
xleft, xright = ax.get_xlim()
ybottom, ytop = ax.get_ylim()
ax.set_aspect(abs((xright-xleft)/(ybottom-ytop))*ratio)

plt.plot(np.log(t), linewidth = 0.9)
plt.rcParams['figure.figsize'] = [7, 5]
apts = np.array([100*np.e])
bpts = np.array(1)
plt.plot(apts, bpts, markersize=7, color='k', marker='s', mfc='none', markeredgewidth=0.5)
plt.ylim(-0.5, 2)

plt.show()
```



1.7. Determine whether any of the functions: (a) $y_1 = \sin 2x$, (b) $y_2(x) = x$, or (c) $y_3(x) = \frac{1}{2}\sin 2x$ is a solution to the initial value problem
$y'' + 4y = 0;\ y(0) = 0,\ y'(0) = 1.$

The easiest way forward is probably just solving the equation and then looking at the offered alternatives for a match.

In [33]:
```
import sympy as sp
from sympy import *
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']
```

In [35]:
```
x = symbols("x")
y = Function("y")(x)
y
```

Out[35]: $y(x)$

In [87]:
```
y.diff()
```

Out[87]: $\frac{d}{dx}y(x)$

In [36]:
```
ode = Eq(y.diff(x,x) + 4*y, 0)
ode
```

Out[36]: $4y(x) + \frac{d^2}{dx^2}y(x) = 0$

In [37]:
```
sol = dsolve(ode, y)
sol
```

Out[37]: $y(x) = C_1 \sin(2x) + C_2 \cos(2x)$

In [38]:
```
checkodesol(ode, sol)
```

Out[38]: (True, 0)

It's pretty confused at the moment, with the various constants clouding the picture. It's time to put in the initial conditions.

In [40]:
```
ics = {y.subs(x,0): 0, y.diff().subs(x,0): 1}
ics
```

Out[40]: {y(0): 0, Subs(Derivative(y(x), x), x, 0): 1}

In [41]:
```
ivp = dsolve(ode, ics=ics)
ivp
```

Out[41]: $y(x) = \frac{\sin(2x)}{2}$

Obviously alternative (c) is the winner here. Alternative (a) demands a quick glance, but since there are no constants to assign, it's relegated to the trash pile.

In [42]:
```
checkodesol(ode, ivp)
```

Out[42]: (True, 0)

1.8. Find the solution to the initial value problem $y' + y = 0;\ y(3) = 2,$ if the general solution to the differential equation is known to be (see Chapter 8) $y(x) = c_1 e^{-x}$, where c_1 is an arbitrary constant.

Most of the work on this one is done. It only remains to find c_1 such that $2 = c_1 e^{-3}$.

In [43]:
```
from sympy import *
const = symbols('const')
eq = Eq(const*exp(-3) -2, 0)
sol = solve(eq)
print(sol)
```

```
[2*exp(3)]
```

In [45]:
```
#testing
tes = 2*exp(3)*exp(-3)
tes
```

Out[45]: 2

Establishing that the value of c_1 is $2\text{sp.exp}(3)$, and the exact solution described is $y(x) = (2\text{sp.exp}(3))e^{-x}$.

1.9. Find a solution to the initial value problem $y'' + 4y = 0;\ y(0) = 0,\ y'(0) = 1$, if the general solution to the differential equation is known to be (see Chapter 9) $y(x) = c_1 \sin 2x + c_2 \cos 2x.$

This is similar to the last problem. A little mental quizzing should bring up the required answers. Starting with the condition of $y(0) = 0$. The factor $c_1 \sin 2x$ would then be zero no matter what value c_1 was assigned. However, to make $\cos 2x$ vanish, c_2 would have to be zero. This leaves the situation as $c_2 = 0$ and c_1 equal to anything. At y' , the one remaining factor has the value of $c_1(2)\cos 2x$, and if $y'(0) = 1$, it means that $c_1 = \frac{1}{2}$. Therefore c_1 should receive the value of $\frac{1}{2}$, making the complete particular solution equal to $\frac{1}{2}\sin 2x$.

1.10. Find a solution to the boundary value problem $y'' + 4y = 0;\ y(\frac{\pi}{8}) = 0\,,\ y(\frac{\pi}{6}) = 1\,,$ if the general solution to the differential equation is $y(x) = c_1 \sin 2x + c_2 \cos 2x.$

In this situation the easiest way to solve it is to just ignore the offering of the general solution. Its veracity can be checked when its natural appearance occurs.

In [16]:
```
import sympy as sp
from sympy import *
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']
```

In [47]:
```
x = symbols("x")
y = Function("y")(x)
y
```

Out[47]: $y(x)$

In [18]:
```
y.diff()
```

Out[18]: $\frac{d}{dx} y(x)$

In [48]:
```
ode = Eq(y.diff(x,x) + 4*y, 0)
ode
```

Out[48]: $4y(x) + \frac{d^2}{dx^2} y(x) = 0$

Now it develops that this ode has been used before.

In [49]:
```
sol = dsolve(ode, y)
sol
```

Out[49]: $y(x) = C_1 \sin(2x) + C_2 \cos(2x)$

Above: The general solution which was proposed in the problem description turns out to be valid.

The next step is to put in the boundary conditions.

In [51]:
```
bcs = {y.subs(x,pi/8): 0, y.diff().subs(x,pi/6): 1}
bcs
```

Out[51]:
```
{y(pi/8): 0, Subs(Derivative(y(x), x), x, pi/6): 1}
```

In [52]:
```
bvp = dsolve(ode, ics=bcs)
bvp
```

Out[52]:
$$y(x) = \left(-\frac{1}{2} + \frac{\sqrt{3}}{2}\right)\sin(2x) + \left(\frac{1}{2} - \frac{\sqrt{3}}{2}\right)\cos(2x)$$

Note: From sympy behavior it appears that the designation of "ics" is required by sympy when incorporating either initial or boundary conditions.

In [53]:
```
checkodesol(ode, bvp)
```

Out[53]:
```
(True, 0)
```

The particular solution is found. It seems clear that scrutinizing and negotiating boundary condition logic would not have made the solution any faster.

1.11. Find a solution to the boundary value problem $y'' + 4y = 0;\ y(0) = 1,\ y(\frac{\pi}{2}) = 2$, if the general solution to the differential equation is known to be $y(x) = c_1 \sin 2x + c_2 \cos 2x$.

This situation is the same as the last problem. The procedure will be to solve the ode for the particular solution without reference to the 'gimme'.

In [26]:
```
import sympy as sp
import matplotlib.pyplot as plt

%config InlineBackend.figure_formats = ['svg']
```

In [55]:
```
x = symbols("x")
y = Function("y")(x)
y
```

Out[55]:
$$y(x)$$

In [28]:
```
y.diff()
```

Out[28]:
$$\frac{d}{dx}y(x)$$

In [56]:
```
ode = Eq(y.diff(x,x) + 4*y, 0)
ode
```

Out[56]:
$$4y(x) + \frac{d^2}{dx^2}y(x) = 0$$

In [57]:
```
sol = dsolve(ode, y)
sol
```

Out[57]:
$$y(x) = C_1 \sin(2x) + C_2 \cos(2x)$$

In [32]:
```
bcs = {y.subs(x, 0): 1, y.subs(x, sp.pi/2): 2}
bcs
```

Out[32]:
```
{y(0): 1, y(pi/2): 2}
```

Above: Here is a new wrinkle. The boundary conditions specify two different values for the solution function, and none for the differential equation.

In [58]:
```
bvp = dsolve(ode, ics=bcs)
bvp
```

Out[58]:
$$y(x) = \left(-\frac{1}{2} + \frac{\sqrt{3}}{2}\right)\sin(2x) + \left(\frac{1}{2} - \frac{\sqrt{3}}{2}\right)\cos(2x)$$

Above: As it turns out, running the above cell results in a big ugly error block. The two simultaneous conditions for the solution equation are conflicting. A prank played by the text authors.

1.12. Determine c_1 and c_2 so that $y(x) = c_1 \sin 2x + c_2 \cos 2x + 1$ will satisfy the conditions $y(\frac{\pi}{8}) = 0$ and $y'(\frac{\pi}{8}) = \sqrt{2}$.

Working here with only the solution equation.

In [87]:
```
import sympy as sp
from sympy import *
x = symbols('x')
c1 = Symbol('c1')
c2 = Symbol('c2')
eq1 = Eq(c1*sin(2*(pi/8)) + c2*cos(2*(pi/8)) + 1,0)
eq2 = Eq(diff(c1*sin(2*(pi/8))) + diff(c2*cos(2*(pi/8))),sqrt(2))
result = solve([eq1,eq2],(c1,c2))
result
```

Out[87]:
```
{c1: -c2 - sqrt(2)}
```

Nicely done, Sympy. The text expresses the result as $c_1 + c_2 = \sqrt{2}$

1.13. Determine c_1 and c_2 so that $y(x) = c_1 e^{2x} + c_2 e^{x} + 2 \sin x$ will satisfy the conditions $y(0) = 0$ and $y'(0) = 1$.

Working here with only the solution equation.

In [79]:
```
import sympy as sp
from sympy import *
x = Symbol('x')
y = Function('y')(x)
c1 = Symbol('c1')
c2 = Symbol('c2')
eq1 = Eq(c1*sp.exp(2*0) + c2*sp.exp(0) + 2*sp.sin(0),0)
```

The above is acceptable for a start. But Sympy is somewhat finicky, and with all the zeros floating around in this problem, if the solve command is given too soon, Sympy will just return an empty bracket. The cell below provides the lhs of eq2.

In [80]:
```
y = c1*exp(2*x) + c2*exp(x) + 2*sin(x)
eqsec = y.diff(x)
eqsec
```

Out[80]: $2c_1e^{2x} + c_2e^{x} + 2\cos(x)$

Adding a rhs to get the complete eq2.

In [81]:
```
eq2 = Eq(eqsec,1)
eq2
```

Out[81]: $2c_1e^{2x} + c_2e^{x} + 2\cos(x) = 1$

Now sympy is ready to solve for c1 and c2.

In [82]:
```
result = solve([eq1,eq2],(c1,c2))
result
```

Out[82]:
```
{c1: -2*cos(x)/(2*exp(2*x) - exp(x)) + 1/(2*exp(2*x) - exp(x)),
 c2: 2*cos(x)/(2*exp(2*x) - exp(x)) - 1/(2*exp(2*x) - exp(x))}
```

What has been delivered has the good info inside, but in form it is considered as a dictionary at the moment. Some massage is required. The descriptions of both constants can be cut and pasted from the above output cell, replacing the colon with an equals sign.

In [83]:
```
c1 = -2*cos(x)/(2*exp(2*x) - exp(x)) + 1/(2*exp(2*x) - exp(x))
c1.subs({x:0})
```

Out[83]: -1

In [84]:
```
c2 = 2*sp.cos(x)/(2*sp.exp(2*x)-sp.exp(x)) - 1/(2*sp.exp(2*x)-sp.exp(x))
c2.subs({x:0})
```

Out[84]: 1

Above: the constants have been determined.