Outcomes

By the end of this notebook, you will be able to...

- Define your own function in Python.
- Use a custom-defined function to create a new data set.
- Adjust function parameters to refine the match between a model and experimental data.

Read in our data

First let's import some data. This code cell is a copy-paste of what we did in CIT 1.2, with the same position, velocity, and acceleration data for a 500-gram cart going up and down a ramp inclined by 6°.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
database = pd.read_excel("https://docs.google.com/spreadsheets/d/1Z4pTRxwg1ZCAc
database.plot.scatter('time','position')
database.plot.scatter('time','velocity')
database.plot.scatter('time','acceleration')
```

Defining a function

The `numpy` library carries lots of standard mathematical functions, but often we need to define our own function as a set of instructions we would like Python to carry out repeatedly. For example, suppose we wanted to explore a constant-acceleration model of the motion of our cart on a ramp. We would reach for our standard equations of motion

$$v(t) = v_i + at \tag{1}$$

$$x(t) = x_i + v_i t + \frac{1}{2}at^2 \tag{2}$$

We can implement these equations as **functions** using Python's `def` structure:

```
def FunctionName(inputs):
  do some math here
  return outputs
```

We give the function a name `FunctionName`, and we can provide any number of `inputs` we want, separated by commas. The section `do some math here` can take up as many lines as we need to carry out what we want `FunctionName` to do. Finally, we `return` all the information that we want `FunctionName` to tell us. Any lines of code that we want **inside the function** get indented by two spaces. This indentation is Python's way of

tracking what code segments are inside the function. This means that the `print()` command is **outside the function**, since it isn't indented.

The code cell below defines a function `vmodel` to carry out the calculation of $v(t) = v_i + at$. What are this function's inputs, and what is its output?

Run the code cell below.

In [ ]:
```
def vmodel(vi,a,t):
  v = vi + a * t
  return v

print('vmodel defined')
```

Try out this function in the code cell below by replacing the `0`s with numerical values and running the code cell. Note that Python uses the variable names and equals signs to track which number needs to be assigned to which input. Repeat this process several times by changing the values you use and checking the result.

This is called **calling the function** that we previously defined.

In [ ]:
```
print( vmodel(vi=0,a=0,t=0) )
```

Add code to the code cell below to carry out the calculation of the position x using the equation above. What inputs will you need in the parentheses? What calculation needs to follow the `x =`? What output should it give after `return`? Run this code cell.

In [ ]:
```
def xmodel():
  x =
  return
```

Now **test** your function by calling it in the code cell below, just like we did with `vmodel` above.

In [ ]:
```
print( xmodel() )
```

In [ ]:
```
#@title Click here to see solution
def xmodel(xi,vi,a,t):
    x = xi + vi * t + 0.5 * a * t**2
    return x

print(xmodel(xi=0,vi=0,a=0,t=0))
```

Generating data from our function

We now have two functions for x and v that we can use to model our cart data. The code cell below creates a new array `database['velocity model']` from our existing array `database['time']`. Add code to print this new array, and run the code cell. Now that

we're passing an **array** as an input for `t` instead of a single number, what sort of output do we get from the function?

Add code to repeat this process to create an array `database['position model']` using the `xmodel` function you defined.

In [ ]:
```
database['velocity model'] = vmodel(vi=0.4,a=-1,t=database['time'])
```

Fitting our model to the data

Now that we have values from our model, we can start to **fit** our model to the experimental data. The code cell below plots **both** our measured velocity values and our model velocity values. Run the code cell below. Note that we're using the `plt.plot()` command instead of the `database.plot()` command because we want these to appear in the same figure. Carry out the following:

1. Which data set is a scatter plot, and which is a line plot?
2. Add axis labels using `xlabel` and `ylabel`.
3. Add code to graph the measured position data and your model's position data. You'll need to display these in a new figure. Copy-paste `plt.figure()` to create a new figure before adding the `plt.plot()` commands. (You don't need anything in the parentheses.)
4. You probably notice that the model data don't entirely match with the measured data. Return to the code cell above and adjust the values of the initial position, initial velocity, and acceleration. Keep adjusting and re-running these two code cells until they do. (And if you get tired of having to go back and forth, feel free to copy-paste one code cell's contents into the other and run the whole thing once!)
5. How is this process of defining your own function more useful than Excel's trendline feature? (And if you find yourself missing your R^2 coefficient, Python can calculate that, too!)

In [ ]:
```
#Generating data from our function
database['velocity model'] = vmodel(vi=0.4,a=-1.1,t=database['time'])
database['position model'] = xmodel(xi=0.039,vi=0.4,a=-1.1,t=database['time'])

#Fitting our Model to the Data. Carry out the following
plt.figure()
plt.scatter(database['time'],database['velocity'])
plt.plot(database['time'],database['velocity model'])

plt.figure()
plt.scatter(database['time'],database['position'])
plt.plot(database['time'],database['position model'])
```