Introduction to Computational Physics (PHYS105)

Lecture 7: Good Practice and Debugging

Carl Gwilliam
(C.Gwilliam@liverpool.ac.uk)
UNIVERSITY OF LIVERPOOL

Attendance code: 277907

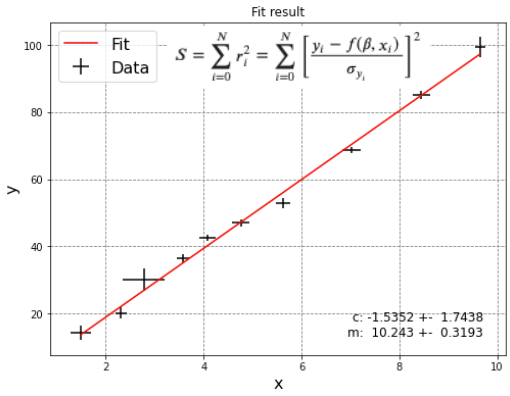
- **RECORD + Attendance Code!!!**
- Morning everyone
  - **Please remember to register your attendance**
- This week we are going to look at good coding practice and how to debug code errors
  - **As always, please interrupt if you have questions**

Lecture 6: Recap

- Last week we started the second part of the course, using python for real physics problems!

- Started with fitting a functional form to data.
  - Read in data from files (using `np.loadtxt`)
    - Plot to check sensible
    - Find initial params
  - Define necessary functions
    - Functional form and its differential
    - Point-by-point residuals
  - Run `least_squares()` from scipy.optimize
    - Check fit succeeds and get fitted params
  - Calculate χ^2 test statistic from the residuals
    - Check fit is good e.g. $0.25 < \chi^2/NDF < 4$
  - Calculate parameter errors
    - Print best-fit parameters and associated error
  - Plot data and fitted function
    - Passing best fit params to functional form

- Encapsulated all this in a reusable function



- You will use this over-and-over in labs
  - Will create your own python module for this in the last week of PHYS105

2

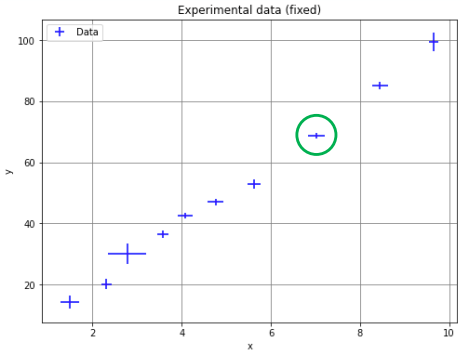
- **First lets recap what we learnt last week, where we started …**
- **In particular, we were fitting a functional form to data to determine the values of the free parameters that best describe it**
- Let's summarise the steps we needed to do this
  - Read in …. using np.loadtxt() + Plot to … and find …
  - **Define … which are the functional form for the model, in our case a straight line, and its differential, which is needed to translate error on x to corresponding y-error**
  - **Also, the function that uses these to calculate array of point-by-point residuals [point + explain] (remember scipy will take care of summing and squaring)**
  - Run the fit using … which is so called because it minimises the sum of the squares of the residuals
    - **This takes the residual function, the initial guess of the parameters and the data + errors as arguments**
    - **… using corresponding properties of the results object**
  - Calculate the chi2 test statistic from the residuals
  - Calculate parameter errors from the Jacobian matrix, which you will learn more about in PHYS108 next semester, and print…
  - **We then pass the best-fit parameters, along with x data, back to our model function to get the fitted y values and plot along with data**
- **This is what LabModule,py does and the result looks something like this**

Lecture 6: Formative problem solutions

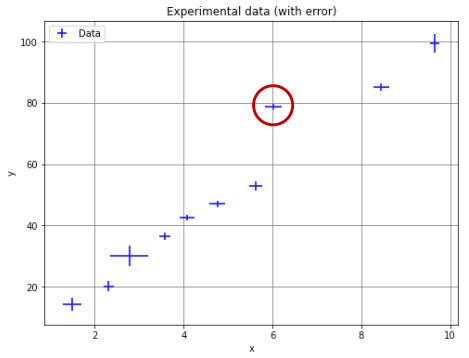
- Exercise 1: Plot the experimental data and fix any errors compared to the table above
  - Plot data read from csv file using `errorbar()`
  - Clear that 8th point is incorrect → fix by updating array value at **index** 7 and replot

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize = (8, 6))
plt.title("Experimental data (with error)")
plt.xlabel("x")
plt.ylabel("y")
plt.errorbar(xdata, ydata, xerr=xerror, yerr=yerror,
             color = 'b', linestyle = '', label = 'Data')
plt.grid(color = 'grey')
plt.legend(loc = 2)
plt.show()
```




```
xdata[7] = 7.02
ydata[7] = 68.8
```



- As usual, we will now look at the solutions to last week's formative problems and feedback on some common issues
- Ex1: Plot the data and fix and issues (don't need scipy yet as not fitting)
  - Take the data that was read from the file and plot using matplotlib as usual, using the error bar function (point)
    - Several people did both plot and errorbar but you only need the latter as errorbar can do everything we need
    - However, when plotting data don't join the markers up → linestyle = "" (point) to remove line
  - Point 8 clearly doesn't lie on the straight line and if we compare to the table in the notebook the file has a mistake for that point in its x and y values (important to check!)
    - Fix by updating the csv file or simply resetting the values [point] in the np array you have read
    - **Some people struggled doing this: remember can access any element of a data structure sing square brackets with the index (remembering it starts form 0 -> 7 for the 8th point) and then, just like any variable, assign a value to this using equals (as we saw in week 2). Need to do this for both the x and y values**
  - **Replotting, it looks much better & consistent with a straight line by eye**
    - **And we fitted it as an example and showed that is the case**

Lecture 6: Formative problem solutions (2)

- Exercise 2: Fit extended data

Read data

```
# Extend data
import numpy as np
xdata2 = np.append(xdata, [10.25, 11.85])
ydata2 = np.append(ydata, [110.5, 122.1])
xerror2 = np.append(xerror, [0.21, 0.41])
yerror2 = np.append(yerror, [3.1, 3.3])
```

Run fit and check it succeeds

```
import numpy as np
from scipy.optimize import least_squares
from scipy.stats import chi2 as stats_chi2
import matplotlib.pyplot as plt
%matplotlib inline

# Set initial params
init_params = [0.0, 10.0]

# Run fit
result = least_squares(minimise, init_params,
                       args=(xdata2, ydata2, xerror2, yerror2))
                                                  New data
# Check fit succeeds
if not result.success or result.status < 1:
    print ("ERROR: Fit failed with message {}".format(result.message))
    print ("Please check the data and inital parameter estimates")
else:
    print ("Fit succeeded")

# Get fitted parameters
final_params = result.x
c = final_params[0]
m = final_params[1]
nparams = len(final_params)
```

Get fitted parameters

Calculate χ^2 and check fit is good

```
# Calculate chi2
chi2_array = result.fun ** 2
chi2 = sum(chi2_array)
npoints = len(xdata)
reduced_chi2 = chi2 / (npoints - nparams)
chi2_prob = stats_chi2.sf(chi2, (npoints - nparams))

# Print chi2
np.set_printoptions(precision = 3)
print("\n=== Fit quality ===")
print("chisq per point = \n",chi2_array)
print("chisq = {:7.5g}, ndf = {}, chisq/NDF = {:7.5g}, chisq prob = {:7.5g}\n".
      format(chi2, npoints-nparams, reduced_chi2, chi2_prob))

if reduced_chi2 < 0.25 or reduced_chi2 > 4:
    print("WARNING: chi2/ndf suspiciously small or large."
          "Please check the data and initial parameter estimates")

if chi2_prob < 0.05:
    print("WARNING: chi2 probability for given degrees of freedom less than 0.05."
          "Please check the data and initial parameter estimates")
```

Calculate param errors and print

```
# Calculate errors
jacobian = result.jac
jacobian2 = np.dot(jacobian.T, jacobian)
determinant = np.linalg.det(jacobian2)

if determinant < 1E-32:
    print(f"Matrix singular (determinant = {determinant}, error calculation failed.")
    param_errors = np.zeros(nparams)
else:
    covariance = np.linalg.inv(jacobian2)
    param_errors = np.sqrt(covariance.diagonal())

print ("c = {:7.5g} +- {:7.5g}".format(final_params[0], param_errors[0]))
print ("m = {:7.5g} +- {:7.5g}".format(final_params[1], param_errors[1]))
```

4

- Ex2: The data was extended by two extra points and we needed to update the fit
- **People struggled with which bits needs to be rerun (and in which order). Lets go through it following the overview from the first slide**
    - Read in the data: in this case rather than updating the CSV file I just used np.append to add the new points to the already read np arrays (point)
        - Called new arrays a different name to make it clear and avoid any conflicts
        - Note: np.append doesn't change original array but returns a new array
    - **Since we are still dealing with a straight line our model functions and the residual function don't change so we don't need to redefine them**
        - **Can also use the same initial parameter guess**
    - **Need to rerun the fit with the new data**
        - **Call least_squares with minimise function, initial param guesses and xdata2 etc (point) in correct order (same as residual func)**
        - **Some people missed this (which is what actually does the fit) → results will then be from the 10-point fit**
        - **Check new fit succeeds and get updated fitted parameters from result.x (point)**
    - Calculate chi2 test statistic from the residuals obtained from result.fun (point), by squaring and summing it, and divide by the NDF (points – params)
        - Then check good fit
    - Calculate the parameter errors from the result.jac
        - Printing the result of the parameters, along with their errors

Lecture 6: Formative problem solutions (3)

- Exercise 2 (cont): Fit extended data

```
# Calculate fitted function values
yfit = straight_line(final_params, xdata2)

# Visualise result
fig = plt.figure(figsize = (8, 6))
plt.title('Fit result')
plt.xlabel('x', fontsize=16)
plt.ylabel('y', fontsize=16)
plt.grid(color = 'grey', linestyle="--")

plt.errorbar(xdata2, ydata2, xerr = xerror2, yerr = yerror2, fmt='k',
             linestyle = '', label = "Data")
plt.plot(xdata2, yfit, color = 'r', linestyle = '-', label = "Fit")

plt.legend(loc = 2, fontsize=16)

text = "c: {:7.5g} +- {:7.5g}\n".format(final_params[0], param_errors[0])
text += "m: {:7.5g} +- {:7.5g}\n".format(final_params[1], param_errors[1])
plt.text(0.95, 0.0, text, transform = fig.axes[0].transAxes, ha = "right",
         va = "bottom", fontsize=12)

plt.show()
```

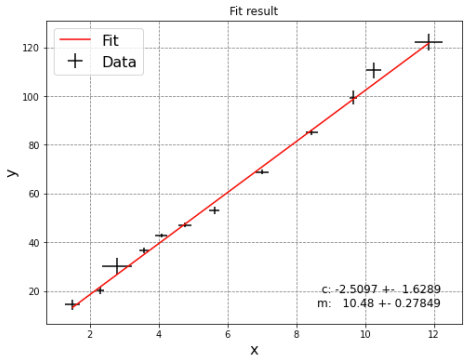
Evaluate best fit y and plot

New data!

```
Fit succeeded

=== Fit quality ===
chisq per point =
 [0.128 0.533 0.387 0.724 1.507 0.016 2.579 1.072 0.156 0.062 2.159 0.006]
chisq =   9.329, ndf = 10, chisq/NDF =  0.9329, chisq prob = 0.50119

c = -2.5097 +-  1.6289
m =   10.48 +- 0.27849
```



5

- Ex2 cont.: Finally, we plot the data (as points) and best-fit function (as line)
  - **Passing the NEW best fit parameters and the NEW x data to the straight_line model function (point)**
  - **Some people passed the old xdata here and yfit will only be 10 elements long → matplotlib will complain about different sized x and y arrays if pass xdata2 when plotting**

* This cell (examples) has everything we need to do the fit and we put it in a function at the end of the notebook

Lecture 6: Formative problem solutions (4)

- Exercise 3: Does straight line still fit the data
  - Yes, as shown by χ^2:
    - χ^2/ndf between ¼ and 4 `chisq/NDF = 1.1661`
    - χ^2 prob greater than 5% `chisq prob = 0.3153`
- Have the gradient or intercept changed significantly?
  - No, values are still compatible within errors
  - Quantify by "Consistency Check" (as in labs)

$$\text{Significance} = \frac{|x_1 - x_2|}{\sqrt{\sigma_{x1}^2 + \sigma_{x_2}^2}} < 3$$

Total error

```
from math import sqrt

def sigma(x1, x2, e1, e2):
    "Calculate number of standard deviations"
    return abs(x1-x2)/sqrt(e1**2 + e2**2)

final_params_10pts = [-1.535, 10.243]
param_errors_10pts = [1.744, 0.319]

csig = sigma(final_params[0], final_params_10pts[0],
             param_errors[0], param_errors_10pts[0])
msig = sigma(final_params[1], final_params_10pts[1],
             param_errors[1], param_errors_10pts[1])

print (f"Significance for c = {csig:.2f} and m = {msig:.2f}")
if csig < 3 and msig < 3:
    print("Values are consistent")
```

```
Significance for c = 0.41 and m = 0.56
Values are consistent
```

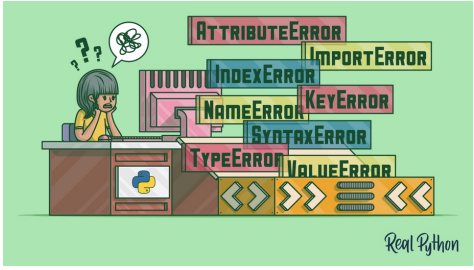
- Now that we have developed our first real physics program, it is time to take a step back …

6

- **Ex3: was about getting you used to thinking about the results of your fit, and doing so in a quantitative manner (which will be needed in labs etc)**
- The first part asked if the straight line still fit the data. To answer that we simply need to check our chi2 test statistics.
  - The chi2/ndf is between ¼ and 4 + the chi2 prob is > 5%, so yes, the straight line fits the data
- The second part asked if the fitted gradient or intercept has changed significantly
  - We can only answer this by considering the uncertainties on the parameters
  - **E.g. the question is 12 compatible with 10 makes no sense w/o the error; with the error we can say e.g. is 12+-2 consistent with 10 (yes) or is 12+-0.1 consistent with 10 (no)**
  - To do this mathematically we use the consistency check you learnt in labs (if you do PHYS106) or was described in the question
  - This says that we take the modulus of the difference between the two values (point) and divide by the total error (adding the errors on each in quadrature)
    - The result tells us how many times the total error they are apart
    - Generally acceptable if < 3 (covers 99.7% of real consistent cases, while 1sigma = 68% and 2sigma = 95%)
  - On the python side, we can write a simple function, that I am going to call sigma, to do this
    - it takes the two values, x1 and x2, along with the two corresponding errors, e1 and e2
    - We subtract the two using the abs function to take the modulus
    - Then we divide by the total error, which we find by adding the two in quad and sqrting
  - **We can call this for both the intercept and gradient, c and m, and in this case both are consistent within 3 sigma (in fact even 1 sigma) so there is no significant change**
- **Note: once you have this function, you can use it each time you need to do such a check in labs!**
  - **Can put it in your lab module in lecture 11**
- **Grades for summative assignment 5, where the average was 82%, were released on canvas on Monday, but 20 people didn't submit anything**
- **Any Questions on last week's formative problems?**

Python errors

- At many points during the course you will have come across issues with your code
  - Where python has given you a big fat error message or you get the wrong result
- We can try to avoid these via good coding practice
  - Using sensible variable names,
  - Writing useful comments,
  - Checking inputs/values, etc
- But no matter how good we are, as we write more complex programs we'll encounter them more often



- There are several different types of error we may encounter
  - Code that is syntactically incorrect and therefore won't parse at all
  - Programs that start but give a run-time error and don't complete
  - Programs that run but produce incorrect results

Harder to debug

- This week, we are going to look in detail at how to understand these errors and debug them

7

- **Good practice: you will see examples of this later in the lecture and in the notebook**
- **Syntactically incorrect, i.e. not valid python, so it doesn't know how to execute it**
- **Run-time error, since we ask python to do something incorrect, and don't complete**
- **Finally: Run without an error message but produce the incorrect result**
- These types of error get harder to the debug as we go down the list

Understanding and debugging error messages

- Real example saw during week 4 PC class

```
1  def rectPrismParams(width, length, height):
2      '''
3      Return volume, surface area and length of edges of
4      rectangular prism given its width,length and height
5      '''
6      vol = width * length * height
7      area = 2 (width * length + width * height + length * height)
8      edge = 4 (width + length + height)
9      return vol, area, edge
10
11 w,l,h = 0.3,0.17,0.25
12 V, A, s = rectPrismParams(w, l, h)
13
14 print(f"Volume = {V:.2f}")
15 print(f"Area   = {A:.2f}")
16 print(f"Edges  = {s:.2f}")
```

- Can turn on line numbers by View → Line Numbers → Show

8

- **Let's take a look at a real example I saw during the week 4 PC class for the exercise calculating the rectangular prism parameters …**
- As a reminder, we wrote a function that takes the width, length and height of the prism
    - Calculates the volume, surface area and length of edges and returns these
- **We call this function with some example parameters and print the returned results**
- The first thing to do to help debugging is to remember to turn on the line numbers in your jupyter notebook
    - By going to the view menu and selecting line numbers and show --> see them here on the left
- If we were to run this …

Understanding and debugging error messages

- Real example saw during week 4 PC class

```
1  def rectPrismParams(width, length, height):
2      '''
3      Return volume, surface area and length of edges of
4      rectangular prism given its width,length and height
5      '''
6      vol = width * length * height
7      area = 2 (width * length + width * height + length * height)
8      edge = 4 (width + length + height)
9      return vol, area, edge
10
11 w,l,h = 0.3,0.17,0.25
12 V, A, s = rectPrismParams(w, l, h)
13
14 print(f"Volume = {V:.2f}")
15 print(f"Area   = {A:.2f}")
16 print(f"Edges  = {s:.2f}")
```

- Can turn on line numbers by View → Line Numbers → Show

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-2185f7667564> in <module>
     10
     11 w,l,h = 0.3,0.17,0.25
---> 12 V, A, s = rectPrismParams(w, l, h)
     13
     14 print(f"Volume = {V:.2f}")

<ipython-input-39-2185f7667564> in rectPrismParams(width, length, height)
      5     '''
      6     vol = width * length * height
----> 7     area = 2 (width * length + width * height + length * height)
      8     edge = 4 (width + length + height)
      9     return vol, area, edge

TypeError: 'int' object is not callable
```

8

- … we would get a scary-looking python error
- But, while it may look scary at first glance, the error message is our friend and tells us how to identify the error
    - We just need to **remember** how to read it …

Understanding and debugging error messages

- Real example saw during week 4 PC class

```
1  def rectPrismParams(width, length, height):
2      '''
3      Return volume, surface area and length of edges of
4      rectangular prism given its width,length and height
5      '''
6      vol = width * length * height
7      area = 2 (width * length + width * height + length * height)
8      edge = 4 (width + length + height)
9      return vol, area, edge
10
11 w,l,h = 0.3,0.17,0.25
12 V, A, s = rectPrismParams(w, l, h)
13
14 print(f"Volume = {V:.2f}")
15 print(f"Area   = {A:.2f}")
16 print(f"Edges  = {s:.2f}")
```

- Can turn on line numbers by View → Line Numbers → Show

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-2185f7667564> in <module>
     10
     11 w,l,h = 0.3,0.17,0.25
---> 12 V, A, s = rectPrismParams(w, l, h)
     13
     14 print(f"Volume = {V:.2f}")

<ipython-input-39-2185f7667564> in rectPrismParams(width, length, height)
      5     '''
      6     vol = width * length * height
----> 7     area = 2 (width * length + width * height + length * height)
      8     edge = 4 (width + length + height)
      9     return vol, area, edge

TypeError: 'int' object is not callable
```

Read from bottom to top

8

- The first thing is that you should always start reading the error message at the bottom
  - Which gives the summary
- And work our way up
  - Which shows how python got to the point of error

Understanding and debugging error messages

- Real example saw during week 4 PC class

```
 1 def rectPrismParams(width, length, height):
 2     '''
 3     Return volume, surface area and length of edges of
 4     rectangular prism given its width,length and height
 5     '''
 6     vol = width * length * height
 7     area = 2 (width * length + width * height + length * height)
 8     edge = 4 (width + length + height)
 9     return vol, area, edge
10
11 w,l,h = 0.3,0.17,0.25
12 V, A, s = rectPrismParams(w, l, h)
13
14 print(f"Volume = {V:.2f}")
15 print(f"Area   = {A:.2f}")
16 print(f"Edges  = {s:.2f}")
```

- Can turn on line numbers by View → Line Numbers → Show

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-2185f7667564> in <module>
     10
     11 w,l,h = 0.3,0.17,0.25
---> 12 V, A, s = rectPrismParams(w, l, h)
     13
     14 print(f"Volume = {V:.2f}")

<ipython-input-39-2185f7667564> in rectPrismParams(width, length, height)
      5     '''
      6     vol = width * length * height
----> 7     area = 2 (width * length + width * height + length * height)
      8     edge = 4 (width + length + height)
      9     return vol, area, edge

TypeError: 'int' object is not callable
```

Read from bottom to top

- The specific type of error that was encountered
  - In this case a "TypeError"
  - Python is telling us the type of the object is incorrect

8

- We starting by looking at the bit highlighted in the red box
- Which tells us what the type or class of error encountered was
  - **We will go over the different types in a bit and in the notebook**
- In this case it is a so-called TypeError
  - i.e. we are doing something that is not supported by the type of variable we have

Understanding and debugging error messages

- Real example saw during week 4 PC class

```
1  def rectPrismParams(width, length, height):
2      '''
3      Return volume, surface area and length of edges of
4      rectangular prism given its width,length and height
5      '''
6      vol = width * length * height
7      area = 2 (width * length + width * height + length * height)
8      edge = 4 (width + length + height)
9      return vol, area, edge
10
11 w,l,h = 0.3,0.17,0.25
12 V, A, s = rectPrismParams(w, l, h)
13
14 print(f"Volume = {V:.2f}")
15 print(f"Area   = {A:.2f}")
16 print(f"Edges  = {s:.2f}")
```

- Can turn on line numbers by View → Line Numbers → Show

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-2185f7667564> in <module>
     10
     11 w,l,h = 0.3,0.17,0.25
---> 12 V, A, s = rectPrismParams(w, l, h)
     13
     14 print(f"Volume = {V:.2f}")

<ipython-input-39-2185f7667564> in rectPrismParams(width, length, height)
      5     '''
      6     vol = width * length * height
----> 7     area = 2 (width * length + width * height + length * height)
      8     edge = 4 (width + length + height)
      9     return vol, area, edge

TypeError: 'int' object is not callable
```

Read from bottom to top

- The specific type of error that was encountered
  - In this case a "TypeError"
  - Python is telling us the type of the object is incorrect
- The error message itself
  - In this case python thinks we are trying to call an integer (like we would do a function)

8

- We then look at the error message itself, which is highlighted in the purple box
- Call integer … which is not possible
  - **Hence the TypeError since we can't call an integer type**

Understanding and debugging error messages

- Real example saw during week 4 PC class

```
1  def rectPrismParams(width, length, height):
2      '''
3      Return volume, surface area and length of edges of
4      rectangular prism given its width,length and height
5      '''
6      vol = width * length * height
7      area = 2 (width * length + width * height + length * height)
8      edge = 4 (width + length + height)
9      return vol, area, edge
10
11 w,l,h = 0.3,0.17,0.25
12 V, A, s = rectPrismParams(w, l, h)
13
14 print(f"Volume = {V:.2f}")
15 print(f"Area   = {A:.2f}")
16 print(f"Edges  = {s:.2f}")
```

- Can turn on line numbers by View → Line Numbers → Show

- The "traceback", showing how the code got to the point of the error through various function calls, going from most recent at the bottom to least recent at the top.
  - The lines indicated by the '--->' are the ones called; the others are for context.
  - In this case it calls `rectPrimsPamas(...)` on L12 & then gives an error in calculating the area on L7

```
TypeError                                 Traceback (most recent call last)
<ipython-input-39-2185f7667564> in <module>
     10
     11 w,l,h = 0.3,0.17,0.25
---> 12 V, A, s = rectPrismParams(w, l, h)
     13
     14 print(f"Volume = {V:.2f}")

<ipython-input-39-2185f7667564> in rectPrismParams(width, length, height)
      5     '''
      6     vol = width * length * height
----> 7     area = 2 (width * length + width * height + length * height)
      8     edge = 4 (width + length + height)
      9     return vol, area, edge

TypeError: 'int' object is not callable
```

Read from bottom to top

- The specific type of error that was encountered
  - In this case a "TypeError"
  - Python is telling us the type of the object is incorrect
- The error message itself
  - In this case python thinks we are tying to call an integer (like we would do a function)

8

- The remaining part of the error message, highlighted in the green box is the so-called traceback …
- In this case it says that the error is on line 7 (point to error)
  - Which is the line calculating the area (as we can see in the code, point)
- It got to that line by calling the rectPrimsParams function on L12 (point)
  - Which is here in the code (point)
- So now we know that
  - Python called our function on L12
  - Executed the code in the function till line 7
  - Tried to call an integer
  - and failed with a TypeError because this is not possible
- **QUESTION: So, can anyone tell me what the error on line 7 is?**

Understanding and debugging error messages

- Real example saw during week 4 PC class

```
1  def rectPrismParams(width, length, height):
2      '''
3      Return volume, surface area and length of edges of
4      rectangular prism given its width,length and height
5      '''
6      vol = width * length * height
7      area = 2 (width * length + width * height + length * height)
8      edge = 4 (width + length + height)
9      return vol, area, edge
10
11 w,l,h = 0.3,0.17,0.25
12 V, A, s = rectPrismParams(w, l, h)
13
14 print(f"Volume = {V:.2f}")
15 print(f"Area   = {A:.2f}")
16 print(f"Edges  = {s:.2f}")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-2185f7667564> in <module>
     10
     11 w,l,h = 0.3,0.17,0.25
---> 12 V, A, s = rectPrismParams(w, l, h)
     13
     14 print(f"Volume = {V:.2f}")

<ipython-input-39-2185f7667564> in rectPrismParams(width, length, height)
      5     '''
      6     vol = width * length * height
----> 7     area = 2 (width * length + width * height + length * height)
      8     edge = 4 (width + length + height)
      9     return vol, area, edge

TypeError: 'int' object is not callable
```

- So, what's the mistake in the code here?
  - We are missing a multiplication sign * between the integer and the brackets!
  - Python sees this as trying to call a function: 2 ()
  - But you can't call an integer, hence the error

- The best way to get familiar with this is practice → let's look at some in a notebook now …

8

- Yes, we are missing a multiplication sign between the integer and the brackets (on both L7 and, in fact, L8)
- While this indicates multiplication in maths it doesn't in python
  - We must explicitly specify this with the asterisk
- If we don't python sees 2 followed by round brackets
  - Which it reads as if we are trying to call the function named 2. But 2 is not a function, it is an integer.
  - So we can't do this -> bang! an error message
- **So, we see how we can use the error message to help us find and fix errors in our code**
  - **Now the only way we can get comfortable with debugging errors this way is to practice**
  - **Let's do this for a few diff. kinds interactively now!**
  - **(You will do more in the notebook)**
- **But first any QUESTIONS on how to understand error messages?**

Summary

- This week we looked at various python errors and how to understand and debug them
- In particular, in the lecture we looked at:
  - Errors that prevent the program running at all
  - Programs that start but don't complete
- Learnt more about how to read python error messages
  - And practiced this with several different error types
- In this week's notebook you will explore errors further:
  - How to reduce the scope for errors with good coding practice
  - Debugging more of the above errors yourself
  - Finally, some programs that run but give the incorrect result!
- You will make coding mistakes (I still make many!)
  - But with practice you can find and fix them quicker!

9

- Any questions on these errors?
- The example errors are available with the rest of the lecture code snippets in this week's CoCalc assignment
  - There is also a version with the errors fixed so you can see what we did
- Any final questions?