# Lecture 4: Debugging

Dr Benjamin J. Morgan[1] and Dr Andrew R. McCluskey[1,2]

[1] *Department of Chemistry, University of Bath, email: b.j.morgan@bath.ac.uk*
[2] *Diamond Light Source, email: andrew.mccluskey@diamond.ac.uk*

October 19, 2019

## Aim

This lecture will introduce *debugging* skills and suggest methods to improve your ability to overcome errors in your code.

## 1 Error messages

Human beings are fallible, therefore code will have bugs. However, there are some tips that we can use to deal with the bugs in our code and make them less of a nuesence.

One of the most powerful tools that is available to us is the quality of the code and libraries that are written in Python. For example, if you `import` the square-root (`sqrt`) function from the basic Python library `math` and give it a list as an argument,

```
from math import sqrt

a = [0, 1, 2, 3]

print(sqrt(a))
```

This will lead to an error. However, this error will come with a *traceback* showing where the error came from and will give some information, at the bottom, about why the error occurred. You should get an error that looks like this,

```
TypeError                                 Traceback (most recent
    ↪ call last)
<ipython-input-14-29539b5bdc2d> in <module>
----> 1 print(sqrt(a))

TypeError: must be real number, not list
```

As we can see, the error that has been thrown is a `TypeError`, and we are given the additional information about the error that: `must be real number, not list`. We can use this to parse the root cause of the problem, a `TypeError`

indicates that *something* is of the wrong *type*, and we are told that this *something* should be a `real number` and not a `list`. It is clear that the arugment being passed to the `sqrt()` function is of *type* `list`, and therefore this is the problem.[1]

Table 1: Some error types that exist in Python.

| Error | Context |
|---|---|
| `IndexError` | Trying to access an invalid index |
| `ModuleNotFoundError` | Trying to import an non-existent module |
| `TypeError` | Performing an action on an inappropriate type |
| `ValueError` | Function argument is an inappropriate type |
| `NameError` | Object with given variable name could not be found |
| `ZeroDivisionError` | Trying to divide something by zero |

**Exercise**

Try and determine the cause of the error messages thrown by the following commands (where a semi-colon indicated a new line):

- `a = [0, 1, 2, 3]; np.square(a[-5])`
- `np.power(2)`
- `mp.array([1, 1, 2, 3, 5, 8, 13])`

If you are not sure what a particular function does search for "numpy" and then the function name online, e.g. "numpy power".

The majority of Python libraries will have helpful error messages such as the one given above (especially commonly used packages such as NumPy, pandas and matplotlib). However, sometimes it is not immediately clear what the error message means, for this it is often necessary to leverage the most important tool in a programmers arsenal – the internet. Consider the following example, we have an array containing the numbers 1 to 9, which we reshape into a `(3, 3)` array and want to obtain the mean of each of the rows (where columns are the first axis and rows the second),

```
import numpy as np

a = np.linspace(1, 9, 9)
a = np.reshape(a, (3, 3))

print(a)

print(np.mean(a, axis=2))
```

[1] Note that the `sqrt` function from the `math` library can only take single values and return the square-root. This NumPy `np.sqrt()` function can however handle objects of type `list` and NumPy arrays.

After running the above code, we will get the following error traceback,

```
IndexError                                Traceback (most recent
    ↪ call last)
<ipython-input-16-d032bc661941> in <module>
----> 1 np.mean(a, axis=2)

~/python3.6/site-packages/numpy/core/fromnumeric.py in mean(a,
    ↪ axis, dtype, out, keepdims)
   3116
   3117     return _methods._mean(a, axis=axis, dtype=dtype,
-> 3118                           out=out, **kwargs)
   3119
   3120

~/python3.6/site-packages/numpy/core/_methods.py in _mean(a,
    ↪ axis, dtype, out, keepdims)
     60
     61     is_float16_result = False
---> 62     rcount = _count_reduce_items(arr, axis)
     63     # Make this warning show up first
     64     if rcount == 0:

~/python3.6/site-packages/numpy/core/_methods.py in
    ↪ _count_reduce_items(arr, axis)
     53     items = 1
     54     for ax in axis:
---> 55         items *= arr.shape[ax]
     56     return items
     57

IndexError: tuple index out of range
```

There is a lot of information here, but we want to focus on just the last line, which reads `IndexError: tuple index out of range`, which is rather opaque. This is where the internet is our friend, if you *copy and paste* this final line into Google[2], you will likely find a lot of hits from a website called **Stack Overflow**. Stack Overflow is a messageboard style website where people can post programming (and mathematics and general computation) questions and others will answer. One of the great things about this website is that it is popular, so nearly all of the questions you could ask have already been answered.

If we put the error message from the above example into our search engine, we shoul be able to get some useful Stack Overflow answers. For example[3], one of the answers suggests *"I suspect you mean to say [0] where you say [1] and [1] where you say [2]. Indexes are 0-based in Python."*, which can immediately lead us to our bug. In the line `print(np.mean(a, axis=2))`, we assigned the row

[2] Or any decent search engine.

[3] https://stackoverflow.com/questions/20296188/indexerror-tuple-index-out-of-range-python


Figure 1: Of course not every question will have an answer. Taken from xkcd (`https://xkcd.com/979/`).

axis to the value 2, however Python counts from 0, to the column axis should be 0, making the row axis 1. If we run the above code again with `axis=1`, we should get the following returned,

```
[2. 5. 8.]
```

It may sound like a rather simple approach to debugging, however, the popularity of Python and the clarity of the error messages mean that *Googling* the problem is often on of the quickest ways to debug a complex issue.

A final note on error messages is one of warning. The nature of Python error messages are as *tracebacks*, this means that the error traces back through the different levels of the library/code and then prints the cause of the error. This means that Python error messages can be **very** long, take the code below (which makes use of the popular plotting library matplotlib) as an example,

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(1, 10, 10)
y = np.linspace(1, 10, 9)

plt.plot(x, y)
plt.show()
```

Running this code will produce an error message that is more than 40 lines long and can be rather worrying the first time that you encounter it. However, it is important not to worry about the size of the error message, but simply to scroll to the bottom of the page and to find the real reason for the error being thrown. In the above example the error that is relevant is `ValueError: x and y must have same first dimension, but have shapes (10,) and (9,)`, which is relatively straight forward to parse. One of the things we are plotting is an array of 10 numbers, while the other is an array of 9 (and therefore it doesn't

make sense to plot them together).

## 2 Raising your own Errors

In your own functions, it is possible *raise* your own errors using the following syntax,

```
# Raising errors

def celsius_to_kelvin(celsius):
    """
    convert from Celsius to Kelvin
    """
    if celsius < -273.15:
        raise ValueError('The temperature was less than '
                         '-273.15 and would produe an '
                         'unphysical temperature of '
                         'less than 0 K')
    else:
        return celsius + 273.15
```

Any of the error type that are outlined in Table 1 may be used in the same way as `ValueError()` is used in the example above. The string that is passed to the `ValueError()` function will be printed to the screen such that the user can better understand the cause of the error.

## 3 Problem

This week we will look at the rotation of a molecule on a surface (such that the $z$ coordinate does not change), for example a water molecule on the surface of some crystal. The ability to rotate a molecule in space is extremely important in computational chemistry, for example in drug discovery a ligand molecule may be rotated with a binding pocket of a protein molecule in order to evaulate the lowest energy interaction and offering insight into ligand design as a result. To help with visualisation of the molecule on a surface, we have provided a **module** on Moodle (`visualisation.py`) that uses the `matplotlib` library to enable visual inspection of the molecule on a surface.

First, you must read in the file given on moodle (`water.txt`) using `np.loadtxt`, then by passing the `x` and `y` arrays to the `visualisation.show(x, y)` function to produce the visualisation. We now turn our focus to the rotation of a molecule (around the origin), Figure 2 shows a graphical description of the transformation required to rotate a point from $(x, y)$ through $\beta$ around the origin to $(x', y')$.

It is shown in Figure 2 that the position of $x$ and $y$ may be described by basic trigonometry as,

$$\begin{aligned} x &= r\cos\alpha, \\ y &= r\sin\alpha. \end{aligned} \tag{1}$$

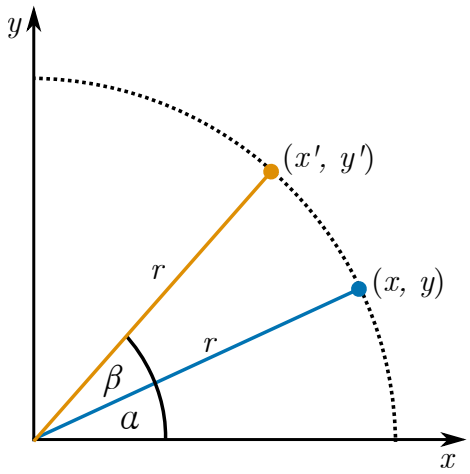


Figure 2: A graphical description of the mathematics underlying the rotation transformation.

While the position of $x'$ and $y'$ can be described as,

$$\begin{aligned} x' &= r\cos\alpha + \beta, \\ y' &= r\sin\alpha + \beta. \end{aligned} \tag{2}$$

Trigonometric identities (which can be found online `http://en.wikipedia.org/wiki/List_of_trigonometric_identities#Angle_sum_and_difference_identities`) may then be used to evaluate $x'$ and $y'$ as follows,

$$\begin{aligned} x' &= r\cos\alpha\cos\beta - r\sin\alpha\sin\beta, \\ y' &= r\sin\alpha\cos\beta - r\cos\alpha\sin\beta. \end{aligned} \tag{3}$$

Finally, by replacing elements of Equation 3 with those from Equation 1 we may obtain straight forward transformations to lead from $(x, y)$ to $(x', y')$,

$$\begin{aligned} x' &= x\cos\beta - y\sin\beta, \\ y' &= y\cos\beta - x\sin\beta. \end{aligned} \tag{4}$$

Using the transformations outlined in Equation 4, you now must create module named `transform` that includes a function called `rotation` that will take three variables `x`, `y`, and `angle`. This function will perform a rotation of `angle` on `x` and `y` to produce `x_new` and `y_new`, which are returned from the function. Use the `visualisation.show` function to observe the rotation of the water molecule.