1.2.8. Exception handling in Python

It is likely that you have raised Exceptions if you have typed all the previous commands of the tutorial. For example, you may have raised an exception if you entered a command with a typo.

Exceptions are raised by different kinds of errors arising when executing Python code. In your own code, you may also catch errors, or define custom error types. You may want to look at the descriptions of the the built-in Exceptions when looking for the right exception type.

1.2.8.1. Exceptions

Exceptions are raised by errors in Python:

```
In [1]: 1/0
---------------------------------------------------------------------------
ZeroDivisionError: integer division or modulo by zero

In [2]: 1 + 'e'
---------------------------------------------------------------------------
TypeError: unsupported operand type(s) for +: 'int' and 'str'

In [3]: d = {1:1, 2:2}

In [4]: d[3]
---------------------------------------------------------------------------
KeyError: 3

In [5]: l = [1, 2, 3]
```

```
In [6]: l[4]
---------------------------------------------------------------------------
IndexError: list index out of range

In [7]: l.foobar
---------------------------------------------------------------------------
AttributeError: 'list' object has no attribute 'foobar'
```

As you can see, there are **different types** of exceptions for different errors.

1.2.8.2. Catching exceptions

1.2.8.2.1. try/except

```
In [10]: while True:
    ....:     try:
    ....:         x = int(raw_input('Please enter a number: '))
    ....:         break
    ....:     except ValueError:
    ....:         print('That was no valid number.  Try again...')
    ....:
Please enter a number: a
That was no valid number.  Try again...
Please enter a number: 1

In [9]: x
Out[9]: 1
```

1.2.8.2.2. try/finally

```
In [10]: try:
    ....:     x = int(raw_input('Please enter a number: '))
    ....: finally:
    ....:     print('Thank you for your input')
    ....:
    ....:
Please enter a number: a
Thank you for your input
---------------------------------------------------------------------------
ValueError: invalid literal for int() with base 10: 'a'
```

Important for resource management (e.g. closing a file)

1.2.8.2.3. Easier to ask for forgiveness than for permission

```
In [11]: def print_sorted(collection):
    ....:     try:
    ....:         collection.sort()
    ....:     except AttributeError:
    ....:         pass
    ....:     print(collection)
    ....:
    ....:

In [12]: print_sorted([1, 3, 2])
[1, 2, 3]

In [13]: print_sorted(set((1, 3, 2)))
set([1, 2, 3])
```

```
In [14]: print_sorted('132')
132
```

1.2.8.3. Raising exceptions

- Capturing and reraising an exception:

```
In [15]: def filter_name(name):
    ....:     try:
    ....:         name = name.encode('ascii')
    ....:     except UnicodeError as e:
    ....:         if name == 'Gaël':
    ....:             print('OK, Gaël')
    ....:         else:
    ....:             raise e
    ....:     return name
    ....:

In [16]: filter_name('Gaël')
OK, Gaël
Out[16]: 'Ga\xc3\xabl'

In [17]: filter_name('Stéfan')
---------------------------------------------------------------------------
      ---
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position 2:
     ordinal not in range(128)
```

- Exceptions to pass messages between parts of the code:

```
In [17]: def achilles_arrow(x):
    ....:     if abs(x - 1) < 1e-3:
    ....:         raise StopIteration
    ....:     x = 1 - (1-x)/2.
    ....:     return x
    ....:

In [18]: x = 0

In [19]: while True:
    ....:     try:
    ....:         x = achilles_arrow(x)
    ....:     except StopIteration:
    ....:         break
    ....:
    ....:

In [20]: x
Out[20]: 0.9990234375
```

Use exceptions to notify certain conditions are met (e.g. StopIteration) or not (e.g. custom error raising)