**Lambda functions**

You've written your very own Python functions using the def keyword, function headers, docstrings and function bodies.

**Lambda functions**

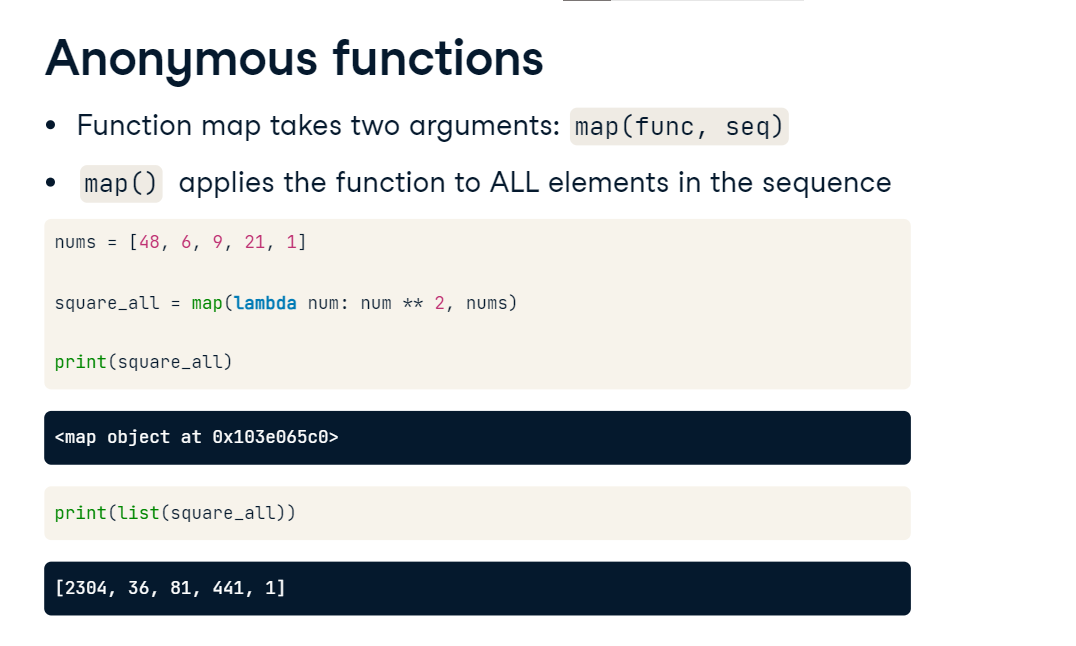
There's a quicker way to write functions on the fly and these are called lambda functions because you use the keyword lambda. Here we re-write our function raise\_to\_power as a lambda function. To do so, after the keyword lambda, we specify the names of the arguments; then we use a colon followed by the expression that specifies what we wish the function to return. Lambda functions allow you to write functions in a quick and potentially dirty way so I wouldn't advise you to use them all the time but there are situations when they can come in very handy.



**Anonymous functions**

For example, check out the map function, which takes two arguments, a function and a sequence such as a list and applies the function over all elements of the sequence. We can pass lambda functions to map without even naming them and in this case we refer to them as anonymous functions. In this example, we use map on a lambda function that squares al

l elements of a list and we'll store the result in square\_all. Printing square\_all reveals that it is actually a map object so to see what it contains we use the function list to turn it into a list and print the results to the shell. As expected, it's a list containing the squares of the elements in the original list!



**Introduction to error handling**

When you use a function incorrectly, it should throw you an error. For example,

**The float() function**

check out the function float that returns a floating point from a number or string, under the condition that the string corresponds to a number.

**Passing an incorrect argument**

When I pass the function float an integer, the corresponding float is returned; similarly if I pass it the string '2.3'. However, if I pass it the string 'hello', Python will throw me an error telling me that it couldn't convert the string to a float. In this case, it threw me a ValueError and there are many types of errors.

**Passing valid arguments**

When we write our own functions, as we have been doing, we may wish to catch specific problems and write specific error messages. Let's check out this user-defined function that computes the square root of a number. It behaves as expected with integers.

**Passing invalid arguments**

What happens if we pass it a string such as 'hello'? Then it throws me an error corresponding to a line of code within the function definition. This error says it was some sort of TypeError but the message may not be particularly useful to a user of our function, so we should endeavor to provide useful error messages for the functions we write.

**Errors and exceptions**

This is an example of an error caught during execution, commonly called exceptions. The main way to catch such exceptions is the try-except clause, in which Python tries to run the code following try and if it can, all is well. If it cannot due to an exception, it runs the code following except.

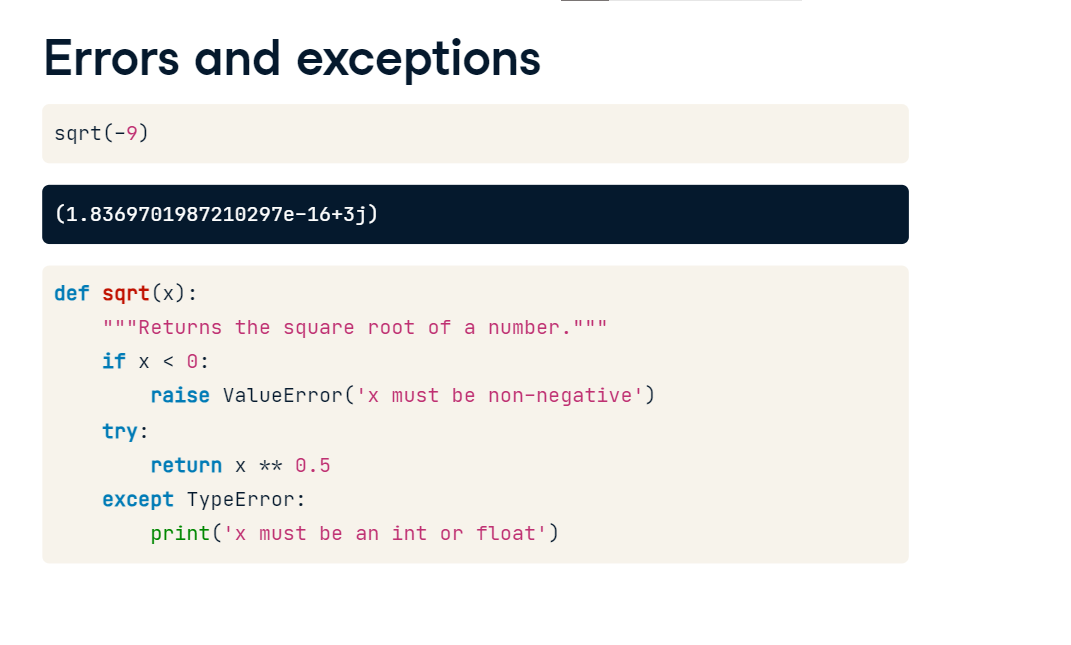


Let's now rewrite our square root function but this time catch any exceptions raised. So here, we try to execute x to the power of zero point five; using except, in the case of an exception, we print 'x must be an int or float'. Now we see that the resulting function behaves well for ints and floats and also prints out what we wanted it to for a string.

**Errors and exceptions**

We may also wish to only catch TypeErrors and let other errors pass through, in which case we would use except TypeError as you can see here. There are many other types of exceptions that can be caught and you can have a look at them in the Python documentation available online.

More often than not, instead of merely printing an error message, we'll want to actually raise an error by using the keyword raise. For example, our square root function does something we may not desire when applied to negative numbers. It actually returns a complex number which we may not want. In fact, let's say that we don't wish our function to work for negative numbers. Then using an if clause, we can raise a ValueError for cases in which the user passes the function a negative number.



**Errors and exceptions**

Let's now see it in action! If we pass our new function a negative number, see it returns the prescribed ValueError! That's enough out of me.

**Bringing it all together**

You're now going to use your hard-earnt skills to write error messages into your DataFrame analyzer that you have been building up in previous chapters:

**Errors and exceptions**

let's say that a user of your function passes your function the name of a column that isn't a column in the DataFrame that they pass it; you'll want to let them know! In the following interactive exercises, you'll write error messages using two methods that you have learned: one, using the try-except syntax that you see here;

**Errors and exceptions**

two: explicitly raising errors using the keyword raise as in this example.