How to Handle Exceptions

How can we handle an exception thrown within a python process

Handling exceptions in Python is really easy. Let's spend some time writing some examples that will cause exceptions. We will start with one of the most common computer science problems: division by zero.

```
1 / 0
# Traceback (most recent call last):
#     File "<string>", line 1, in <fragment>
# ZeroDivisionError: integer division or modulo by zero
```

If you think back to elementary math class, you will recall that you cannot divide by zero. In Python, this operation will cause an error, as you can see in the first half of the example. To catch the error, we wrap the operation with a **try/except** statement.

```
try:
    1 / 0
except ZeroDivisionError:
    print("You cannot divide by zero!")

# You cannot divide by zero!
```

Bare Excepts

Here's another way to catch the error:

```
try:
    1 / 0
except:
    print("You cannot divide by zero!")
```

This is **not** recommended! In Python, this is known as a **bare except**, which means it will catch any and all exceptions. The reason this is not recommended is that you don't know which exception you are catching. When you have something like **except ZeroDivisionError**, you are obviously trying to catch a division by zero error. In the code above, you cannot tell what you are trying to catch.

Let's take a look at a couple of other examples.

```
my_dict = {"a":1, "b":2, "c":3}
try:
    value = my_dict["d"]
except KeyError:
    print("That key does not exist!")

my_list = [1, 2, 3, 4, 5]
try:
    my_list[6]
except IndexError:
    print("That index is not in the list!")
```

In the first example, we create a 3-element dictionary. Then we try to access a key that is not in the dictionary. Because the key is not in the dictionary, it raises a **KeyError**, which we catch. The second example shows a list that is 5 items in length. We try to grab the 7th item from the index. Remember, Python lists are zero-based, so when you say [6], you're asking for the 7th item. Anyway, because there are only 5 items, it raises an **IndexError**, which we also catch.

You can also catch multiple exceptions with a single statement. There are a couple of different ways to do this. Let's take a look:

```
my_dict = {"a":1, "b":2, "c":3}
try:
    value = my_dict["d"]
except IndexError:
    print("This index does not exist!")
```

```
except KeyError:
    print("This key is not in the dictionary!")

except:
    print("Some other error occurred!")
```

This is a fairly standard way to catch multiple exceptions. First we try to access a key that doesn't exist in the dictionary. The try/except checks to see if you are catching a KeyError, which you are in the second **except** statement. You will also note that we have a bare except at the end. This is usually not recommended, but you'll probably see it from time to time, so it's good to know about it. Also note that most of the time, you won't need to wrap a block of code in multiple **except** handlers. You normally just need to wrap it in one.

Here's another way to catch multiple exceptions:

```
my_dict = {"a":1, "b":2, "c":3}

try:
    value = my_dict["d"]
except (IndexError, KeyError):
    print("An IndexError or KeyError occurred!")
```

Notice that in this example, we are putting the errors that we want to catch inside of parentheses. The problem with this method is that it's hard to tell which error has actually occurred, so the previous example is recommended.

Most of the time when an exception occurs, you will need to alert the user by printing to the screen or logging the message. Depending on the severity of the error, you may need to exit your program. Sometimes you will need to clean up before you exit the program. For example, if you have opened a database connection, you will want to close it before you exit your program or you may end up leaving connections open. Another example is closing a file handle that you have been writing to. You will learn more about file handling in the next chapter. But first, we need to learn how to clean up after ourselves. This is facilitated with the **finally** statement.