19.3. When to use try/except

The reason to use try/except is when you have a code block to execute that will sometimes run correctly and sometimes not, depending on conditions you can't foresee at the time you're writing the code.

For example, when you are running code that fetches data from a website, you may run the code when you don't have a network connection or when the external website is temporarily not responding. If your program can still do something useful in those situations, you would like to handle the exception and have the rest of your code execute.

As another example, suppose you have fetched some nested data from a website into a dictionary d. When you try to extract specific elements, some may be missing: d may not include a particular key, for example. If you anticipate a particular key potentially not being present, you could write an if..else check to take care of it.

```
if somekey in d:
    # it's there; extract the data
    extract_data(d)
else:
    skip_this_one(d)
```

However, if you're extracting lots of different data, it can get tedious to check for all of them. You can wrap all the data extraction in a try/except.

```
try:
    extract_data(d)
except:
    skip_this_one(d)
```

It's considered poor practice to catch all exceptions this way. Instead, python provides a mechanism to specify just certain kinds of exceptions that you'll catch (for example, just catching exceptions of type KeyError, which happens when a key is missing from a dictionary.

```
try:
    extract_data(d)
except KeyError:
    skip_this_one(d)
```

We won't go into more details of exception handling in this introductory course. Check out the official python tutorial section on error handling if you're interested.

You have attempted 1 of 1 activities on this page

19.1. What is an exception?">
hat is an exception?">
19.4. Standard Exceptions">
✔ Completed. Well Done!
19.4. Standard Exceptions">Next Section - 19.4. Standard Exceptions

| Back to top