Module Import Checking

Python has a coding style that is known as EAFP: Easier to ask for forgiveness than permission. What this means is that it's often easier to just assume that something exists (like a key in a dict) and catch an exception if we're wrong. You saw this in our previous chapter where we would attempt to import a module and we caught the **ImportError** if it didn't exist. What if we wanted to check and see if a module could be imported rather than just guessing? You can do that with importlib! Let's take a look:

```
import importlib.util

def check_module(module_name):
    """
    Checks if module can be imported without actually
    importing it
    """
    module_spec = importlib.util.find_spec(module_name)
    if module_spec is None:
        print('Module: {} not found'.format(module_name))
        return None
    else:
        print('Module: {} can be imported!'.format(module_name))
        return module_spec


def import_module_from_spec(module_spec):
    """
    Import the module via the passed in module specification
    Returns the newly imported module
    """
    module = importlib.util.module_from_spec(module_spec)
    module_spec.loader.exec_module(module)
    return module

if __name__ == '__main__':
    module_spec = check_module('fake_module')
    module_spec = check_module('collections')
    if module_spec:
        module = import_module_from_spec(module_spec)
        print(dir(module))
```

Here we import a submodule of importlib called **util**. The **check_module** code has the first piece of magic that we want to look at. In it we call the **find_spec** function against the module string that we passed in. First we pass in a fake name and then we pass in a real name of a Python module. If you run this code, you will see that when you pass in a module name that is not installed, the **find_spec** function will return **None** and our code will print out that the module was not found. If it was found, then we will return the module specification.

We can take that module specification and use it to actually import the module. Or you could just pass the string to the **import_module** function that we learned about in the previous section. But we already covered that so let's learn how to use the module specification. Take a look at the **import_module_from_spec** function in the code above. It accepts the module specification that was returned by **check_module**. We then pass that into importlib's **module_from_spec** function, which returns the import module. Python's documentation recommends executing the module after importing it, so that's what we do next with the **exec_module** function. Finally we return the module and run Python's **dir** against it to make sure it's the module we expect.