Caching with functools.lru_cache

The functools module provides a handy decorator called **lru_cache**. Note that it was added in Python 3.2. According to the documentation, it will "wrap a function with a memoizing callable that saves up to the maxsize most recent calls". In other words, it's a decorator that adds caching to the function it decorates. Let's write a quick function based on the example from the functools documentation that will grab various web pages. In this case, we'll be grabbing pages from the Python documentation site.

```
import urllib.error
import urllib.request

from functools import lru_cache


@lru_cache(maxsize=24)
def get_webpage(module):
    """
    Gets the specified Python module web page
    """
    webpage = "https://docs.python.org/3/library/{}.html".format(module)
    try:
        with urllib.request.urlopen(webpage) as request:
            return request.read()
    except urllib.error.HTTPError:
        return None

if __name__ == '__main__':
    modules = ['functools', 'collections', 'os', 'sys']
    for module in modules:
        page = get_webpage(module)
    if page:
        print("{} module page found".format(module))
```

In the code above, we decorate our **get_webpage** function with **lru_cache** and set its max size to 24 calls. Then we set up a webpage string variable and pass in which module we want our function to fetch. I found that this works best if

you run it in a Python interpreter, such as IDLE. This allows you to run the loop a couple of times against the function. What you will quickly see is that the first time it runs the code, the output is printed our relatively slowly. But if you run it again in the same session, you will see that it prints immediately which demonstrates that the lru_cache has cached the calls correctly. Give this a try in your own interpreter instance to see the results for yourself.

There is also a **typed** parameter that we can pass to the decorator. It is a Boolean that tells the decorator to cache arguments of different types separately if typed is set to **True**.