instead of

```
from string import *
```

prevents naming collisions.

ndard library A library is a collection of software used as tools in the development of other software. The standard library of a programming language is the set of such tools that are distributed with the core programming language. Python comes with an extensive standard library.

12.11 Exercises

1. Open help for the `calendar` module.
   (a) Try the following:

```
import calendar
cal = calendar.TextCalendar()        # Create an instance
cal.pryear(2012)                     # What happens here?
```

   (b) Observe that the week starts on Monday. An adventurous CompSci student believes that it is better mental chunking to have his week start on Thursday, because then there are only two working days to the weekend, and every week has a break in the middle. Read the documentation for TextCalendar, and see how you can help him print a calendar that suits his needs.
   (c) Find a function to print just the month in which your birthday occurs this year.
   (d) Try this:

```
d = calendar.LocaleTextCalendar(6, "SPANISH")
d.pryear(2012)
```

   Try a few other languages, including one that doesn't work, and see what happens.
   (e) Experiment with `calendar.isleap`. What does it expect as an argument? What does it return as a result? What kind of a function is this?

   Make detailed notes about what you learned from these exercises.
2. Open help for the `math` module.
   (a) How many functions are in the `math` module?
   (b) What does `math.ceil` do? What about `math.floor`? (*hint:* both `floor` and `ceil` expect floating point arguments.)
   (c) Describe how we have been computing the same value as `math.sqrt` without using the `math` module.
   (d) What are the two data constants in the `math` module?

Record detailed notes of your investigation in this exercise.

3. Investigate the `copy` module. What does `deepcopy` do? In which exercises from last chapter would `deepcopy` have come in handy?

4. Create a module named `mymodule1.py`. Add attributes `myage` set to your current age, and `year` set to the current year. Create another module named `mymodule2.py`. Add attributes `myage` set to 0, and `year` set to the year you were born. Now create a file named `namespace_test.py`. Import both of the modules above and write the following statement:

```
print( (mymodule2.myage - mymodule1.myage) ==
       (mymodule2.year - mymodule1.year)   )
```

When you will run `namespace_test.py` you will see either `True` or `False` as output depending on whether or not you've already had your birthday this year.

What this example illustrates is that out different modules can both have attributes named `myage` and `year`. Because they're in different namespaces, they don't clash with one another. When we write `namespace_test.py`, we fully qualify exactly which variable `year` or `myage` we are referring to.

5. Add the following statement to `mymodule1.py`, `mymodule2.py`, and `namespace_test.py` from the previous exercise:

```
print("My name is", __name__)
```

Run `namespace_test.py`. What happens? Why? Now add the following to the bottom of `mymodule1.py`:

```
if __name__ == "__main__":
    print("This won't run if I'm  imported.")
```

Run `mymodule1.py` and `namespace_test.py` again. In which case do you see the new print statement?

6. In a Python shell / interactive interpreter, try the following:

```
>>> import this
```

What does Tim Peters have to say about namespaces?

7. Give the Python interpreter's response to each of the following from a continuous interpreter session:

```
>>> s = "If we took the bones out, it wouldn't be crunchy, would it?"
>>> s.split()
>>> type(s.split())
>>> s.split("o")
>>> s.split("i")
>>> "0".join(s.split("o"))
```

Be sure you understand why you get each result. Then apply what you have learned to fill in the body of the function below using the `split` and `join` methods of `str` objects:

```
def myreplace(old, new, s):
    """ Replace all occurrences of old with new in s. """
    ...


test(myreplace(",", ";", "this, that, and some other thing") ==
                         "this; that; and some other thing")
test(myreplace(" ", "**",
                "Words will now      be  separated by stars.") ==
                "Words**will**now**be**separated**by**stars.")
```

Your solution should pass the tests.

8. Create a module named `wordtools.py` with our test scaffolding in place.

   Now add functions to these tests pass:

```
test(cleanword("what?") == "what")
test(cleanword("'now!'") == "now")
test(cleanword("?+='w-o-r-d!,@$()'") ==  "word")

test(has_dashdash("distance--but"))
test(not has_dashdash("several"))
test(has_dashdash("spoke--"))
test(has_dashdash("distance--but"))
test(not has_dashdash("-yo-yo-"))

test(extract_words("Now is the time!  'Now', is the time? Yes, now.") ==
      ['now','is','the','time','now','is','the','time','yes','now'])
test(extract_words("she tried to curtsey as she spoke--fancy") ==
      ['she','tried','to','curtsey','as','she','spoke','fancy'])

test(wordcount("now", ["now","is","time","is","now","is","is"]) == 2)
test(wordcount("is", ["now","is","time","is","now","the","is"]) == 3)
test(wordcount("time", ["now","is","time","is","now","is","is"]) == 1)
test(wordcount("frog", ["now","is","time","is","now","is","is"]) == 0)

test(wordset(["now", "is", "time", "is", "now", "is", "is"]) ==
      ["is", "now", "time"])
test(wordset(["I", "a", "a", "is", "a", "is", "I", "am"]) ==
      ["I", "a", "am", "is"])
test(wordset(["or", "a", "am", "is", "are", "be", "but", "am"]) ==
      ["a", "am", "are", "be", "but", "is", "or"])

test(longestword(["a", "apple", "pear", "grape"]) == 5)
test(longestword(["a", "am", "I", "be"]) == 2)
test(longestword(["this","supercalifragilisticexpialidocious"]) == 34)
test(longestword([ ]) == 0)
```

   Save this module so you can use the tools it contains in future programs.