1) What is the difference between enclosing a list comprehension in square brackets and parentheses?

Ans: List comprehensions in square brackets produce the result list all at once in memory. When they are enclosed in parentheses instead, they are actually generator expressions—they have a similar meaning but do not produce the result list all at once. Instead, generator expressions return a generator object, which yields one item in the result at a time when used in an iteration context.

2) What is the relationship between generators and iterators?

Ans: Generators are iterable objects that support the iteration protocol automatically— they have an iterator with a __next__ method (next in 2.X) that repeatedly advances to the next item in a series of results and raises an exception at the end of the series. In Python, we can code generator functions with def and yield, generator expressions with parenthesized comprehensions, and generator objects with classes that define a special method named __iter__ (discussed later in the book).

3) What are the signs that a function is a generator function?

Ans: A generator function has a yield statement somewhere in its code. Generator functions are otherwise identical to normal functions syntactically, but they are compiled specially by Python so as to return an iterable generator object when called. That object retains state and code location between values.

4) What is the purpose of a yield statement?

Ans: When present, this statement makes Python compile the function specially as a generator; when called, the function returns a generator object that supports the iteration protocol. When the yield statement is run, it sends a result back to the caller and suspends the function's state; the function can then be resumed after the last yield statement, in response to a next built-in or __next__ method call issued by the caller. In more advanced roles, the generator send method similarly resumes the generator, but can also pass a

value that shows up as the yield expression's value. Generator functions may also have a return statement, which terminates the generator.

5) What is the relationship between map calls and list comprehensions? Make a comparison and contrast between the two.

Ans: The map call is similar to a list comprehension—both produce a series of values, by collecting the results of applying an operation to each item in a sequence or other iterable, one item at a time. The primary difference is that map applies a function call to each item, and list comprehensions apply arbitrary expressions. Because of this, list comprehensions are more general; they can apply a function call expression like map, but map requires a function to apply other kinds of expressions. List comprehensions also support extended syntax such as nested for loops and if clauses that subsume the filter built-in. In Python 3.X, map also differs in that it produces a generator of values; the list comprehension materializes the result list in memory all at once. In 2.X, both tools create result lists.