需要特别注意的是函数入口和出口中 Python 类型和 SQL 类型的映射。比如，在入口参数中，SQL 中的 boolean 转换为 Python 的 bool 对象，int 转换为 Python 的 int 对象，数组转换为 Python 的 list，等等；在函数返回的时候，PL/Python 也会把 Python 对象转换为各种 SQL 类型。我们可以在 PostgreSQL 的官方文档中看到所有类型的映射细节。以前面提到的 pyinc() 函数和表 test_tbl 为例，对于下面的 SQL 查询，执行器会读取 test_tbl 中的每个元组中的 a 那一列，a 在 test_tbl 表中定义为 int 格式，那么执行器在调用 Python 解释器来执行 pyinc() 函数中的 Python 代码之前，会把 a 这一列的数据转换成 Python 的 int 对象。

```
postgres=# select pyinc(a) from test_tbl;
```

PL/Python 函数也支持 set returning 函数。可以把 set returning 函数简单地理解为一个迭代器。函数的一次调用会返回一组数据，最终查询输出会把每一组数据展开或者输出为多行，因此这种函数在性能方面会有一定的优势。下面是一个简单的 set returning 函数例子：

```
CREATE OR REPLACE FUNCTION pyreturnsetofint(num int) RETURNS setof int AS $BODY$
return [x for x in range(num)]
$BODY$ LANGUAGE plpythonu;
```

8.1.6 PL/Python 函数中的数据共享

PL/Python 支持函数级别和会话级别的数据共享，即允许在函数调用时候把一些数据保存在内存，供这个函数调用或者这个会话中的其他函数使用。函数级别的数据共享是通过 SD 词典实现的，而会话级别的数据共享是通过 GD 词典实现的。使用这两个词典有时候会大幅提升性能。我们以下面的函数为例解释 SD 和 GD 的使用。

```
CREATE OR REPLACE FUNCTION pylog(a integer, b integer) RETURNS double precision AS $$
  import math
  return math.log(a, b)
  $$ LANGUAGE plpythonu;

select pylog(a, b) from tbl;
```

这个函数的功能是对于 tbl 表中的每一行计算一个 log 函数，其中用到了 math 这个 Python 包。Greenplum 执行器在处理 tbl 表的每一行时会运行这个函数，而每次运行都要导入 math 这个 Python 包，这样会影响性能。但是如果把函数修改为下面这样：

```
CREATE OR REPLACE FUNCTION pylog(a integer, b integer) RETURNS double precision AS $$
If 'math' not in GD:
  import math
  GD['math'] = math
return GD['math'].log(a, b)
$$ LANGUAGE plpythonu;
```