Top of Form



Bottom of Form

Top of Form



# Python 2: Getting More Out of Python Lesson 11, Project 1

Handed in: 27 Mar 2015 04:53:38PM Graded: 30 Mar 2015 02:54:28PM

**Here are your instructions:**

Modify the classFactory.py source code so that the DataRow class returned by the build\_row function has another method:

retrieve(self, curs, condition=None)

self is (as usual) the instance whose method is being called, curs is a database cursor on an existing database connection, and condition (if present) is a string of condition(s) which must be true of all received rows.

The retrieve method should be a generator, yielding successive rows of the result set until it is completely exhausted. Each row should be a next object of type DataRow.

##### Your Comment:

Pat,

Thanks for all of your insight with this project. I'm still not sure I'm doing it correctly, but I'm pretty close I think. Have a great weekend!

##### Items Handed In

* [Open Project Handed In](https://students.oreillyschool.com/student/project/?/.handin/160-7909-2/com.ost.jwoloson.160.7909.2.DatabaseHints_Homework.zip)

### Overall Comments:

Hi Jason,

This is pretty close, except the idea is not to yield a row of data, but a dataRow object.

The fundamental issue here is that the build\_row() function does not have a retrieve () method. However, the object it returns (the dataRow class) does contain a retrieve() method. build\_row( ) returns a custom version of DataRow (not an instance) comfortable with a particular table and its columns.

The fact that the retrieve( ) method the project wants you to add to DataRow (inside build\_row) has 'self' as its first argument tells us that the class designer envisions invoking the retrieve method not on DataRow-the-class, but on individual instances of that class.

Some future project might want to implement .insert( ) to take a row of data and insert it into the DataRow table. A specific instance would be needed, to have any data to start with. We may think of .retrieve and .insert as a pair of functions that work on DataRow \*instances\* (those that wrap a row of data, either coming or going).

How does one instance a DataRow? By feeding it data, i.e.

self.a = A([53,'Zorro', 'Zebra', 340])

... is triggering A.\_\_init\_\_( ) where A is a custom version of DataRow.

What gets returned and named 'a' (no quotes) by this statement:

a = self.a.retrieve(cursor, "id > 51")

is an iterator, i.e. next(a) will make retrieve( ) execute until it encounters a yield, at which point a first DataRow is handed to the caller, a DataRow wrapping data from the table:

curs.execute(thequery) # run SQL select

for r in fetchall():

yield DataRow( r )

next(a) again would advance to the next 'yield' which is the same one just around the loop one more time, so new data.

def test\_retrieve(self):

A = build\_row('animal', 'id name family weight')

self.a = A([53,'Zorro', 'Zebra', 340])

a = self.a.retrieve(cursor, "id > 51")

for v in a:

print( repr(v) )

... would be a way to print the resulting DataRows to exhaustion.

However, printing to console is not the best way to design unittests. Better is to think of assertions that will result in pass or fail.

Could you please have another look?

-Pat

### Grade:

Try Again

Your instructor would like you to work on this project. Be sure to hand in your corrections!

[Take Me Back](ostreturn:)   [Previous Attempt](https://students.oreillyschool.com/student/viewassignment.php?entryid=7909&attempt=1&graded=1)

© 2014, O'Reilly Media, Inc. All rights reserved.

Bottom of Form