# Python eval(), apply(), explicit type coercion and a sorting function

**Assignment 1 for CSC 310 and CSC 580, Spring, 2009**
**Dr. Dale E. Parson, http://faculty.kutztown.edu/parson**
**CSC 580 is due 11:59 PM on February 2, CSC 310 is due 11:59 PM on Feb. 9**

Programming assignments for CSC 310 and CSC 580 will diverge after this assignment. The goals in this first assignment are to initiate students into programming with the Python 2.4 or higher 2.x interpreter by writing and invoking a sort function, including run-time interpretation using eval(), apply() and explicit dynamic type coercion, along with gmake-driven testing.

Perform the following steps to copy and inspect my initial code handout.

```
cp -pr ~parson/ProcLang/mysort ~/ProcLang/mysort
cd ~/ProcLang/mysort
```

Student changes go into files mysort.py and makefile. Search for the string "STUDENT" to find student code requirements.

File **mysort.py** is a modified copy of the initial handout mergesort.py that requires students to complete some coding. This program has the following conventions on its command line arguments (taken from mysort.py):

```
def main():
    ''''''
    Function:        main

        main is a test driver that uses the name of a function supplied
        on the command line to find a sorting function, and applies
        that function to an infix comparison operator and a list of
        typed values, also supplied on the command line.
        It prints the unsorted and then the sorted list.
    Parameters:
        argv[1] is the name of a sort function that takes two parameters:
        a list of values to be sorted, and an infix comparison operator,
        in the form of a string, that gives a boolean result and determines
        the sort direction.
        For example "<=" sorts in ascending sequence and ">=" sorts in
        descending sequence.

        argv[2] is the string representation of the comparison operator.

        argv[3] is the type name of the value list for coercion.

        argv[4] .. argv[argc-1] is an array of type-compatible values
        to be sorted.
    Return value:     SUCCESS (0) or PARMERR on parameter error
```

Student requirements 1 through 4 appear in mysort.py, following “STUDENT” tagged comments, and student requirement 5 appears in the makefile.

Requirement 1 entails using eval() in combination with a type name in argv[3] and a value in argv[4] or higher, in order to coerce a string command line argument to a typed object whose type is named by argv[3].

Requirement 2 requires using eval() to map the string in argv[1] to a sort function object.

Requirement 3 requires using apply() to invoke the sort function of step 2 on its arguments.

Requirement 4 requires writing a sort function (other than mergesort) that is compatible with the signature of mergesort as given in mysort.py. Step 3 invokes this sort function.

Requirement 5 requires adding and successfully executing ascending and descending sort tests on different type parameters for the sort function of step 4. The makefile documents this requirement.

Below is an example test run for mergesort, before addition of student tests. We will inspect the reference files in class.

```
-bash-3.00$ gmake clean test
/bin/rm -f *.o *.class .jar core *.exe *.obj *.pyc
/bin/rm -f *.out *.dif
chmod +x mysort.py
./mysort.py mergesort "<=" int > mysort.py.out
./mysort.py mergesort "<=" int 11 >> mysort.py.out
./mysort.py mergesort "<=" int 11 -11 >> mysort.py.out
./mysort.py mergesort "<=" int 11 -11 0 >> mysort.py.out
./mysort.py mergesort "<=" int 999 -999 123 -321 0 9 0 4 >> mysort.py.out
./mysort.py mergesort "<=" int 999 123 -321 0 -999 9 0 4 -7 >> mysort.py.out
./mysort.py mergesort "<=" int -3 -2 -1 0 1 2 3 >> mysort.py.out
./mysort.py mergesort "<=" int 3 2 1 zero -10 -20 >> mysort.py.out
./mysort.py mergesort "<=" str 3 2 1 zero -10 -20 >> mysort.py.out
./mysort.py mergesort "<=" float 3 2 1 zero -10 -20 >> mysort.py.out
diff mysort.py.out mysort.py.ref > mysort.py.dif
./mysort.py mergesort ">=" int > mysort.py2.out
./mysort.py mergesort ">=" int 11 >> mysort.py2.out
./mysort.py mergesort ">=" int 11 -11 >> mysort.py2.out
./mysort.py mergesort ">=" int 11 -11 0 >> mysort.py2.out
./mysort.py mergesort ">=" int 999 -999 123 -321 0 9 0 4 >> mysort.py2.out
./mysort.py mergesort ">=" int 999 123 -321 0 -999 9 0 4 -7 >> mysort.py2.out
./mysort.py mergesort ">=" int -3 -2 -1 0 1 2 3 >> mysort.py2.out
./mysort.py mergesort ">=" str -3 -2 -1 0 1 2 3 >> mysort.py2.out
./mysort.py mergesort ">=" float -3 -2 -1 0 1 2 3 >> mysort.py2.out
diff mysort.py2.out mysort.py2.ref > mysort.py2.dif
```

Use **gmake turnitin** from within the mysort/ directory to turn it in.