2to3 tool

Python 2.x 2.6–2.7 , Python 3.x 3.0–3.6

Examples

Basic Usage

Consider the following Python2.x code. Save the file as example.py

```
def greet(name):
    print "Hello, {0}!".format(name)
print "What's your name?"
name = raw_input()
greet(name)
```

In the above file, there are several incompatible items. The raw_input() method has been replaced with input() in Python3.x and print is no longer a statement, but a function. This code can be converted to Python 3.x code using the 2to3 tool.

```
$ 2to3 example.py
```

Running the above code will output the diff against the original source file as shown below.

```
RefactoringTool: Skipping implicit fixer: buffer
RefactoringTool: Skipping implicit fixer: idioms
RefactoringTool: Skipping implicit fixer: set_literal
RefactoringTool: Skipping implicit fixer: ws_comma
RefactoringTool: Refactored example.py
--- example.py    (original)
+++ example.py    (refactored)
@@ -1,5 +1,5 @@
 def greet(name):
-    print "Hello, {0}!".format(name)
-print "What's your name?"
-name = raw_input()
+    print("Hello, {0}!".format(name))
+print("What's your name?")
+name = input()
 greet(name)
RefactoringTool: Files that need to be modified:
RefactoringTool: example.py
```

The modifications can be written back to the source file using the -w flag. A backup of the original file called example.py.bak is created, unless the -n flag is given.

```
$ 2to3 -w example.py
```

Now the example.py file has been converted from Python 2.x to Python 3.x code.

Afterwards, example.py contains the following valid Python3.x code:

```
def greet(name):
    print("Hello, {0}!".format(name))
print("What's your name?")
name = input()
greet(name)
```

Syntax

```
$ 2to3 [-options] filename / directory_name
```

Parameters

Parameter	Description
filename / directory_name	2to3 accepts a list of files or directories which is to be transformed as its argument. The directories are recursively traversed for Python sources.
Option	**Option Description**
-f FIX, --fix=FIX	Specify transformations to be applied; default: all. List available transformations with --list-fixes

Parameter	Description
-j PROCESSES, –processes=PROCESSES	Run 2to3 concurrently
-x NOFIX, –nofix=NOFIX	Exclude a transformation
-l, –list-fixes	List available transformations
-p, –print-function	Change the grammar so that `print()` is considered a function
-v, –verbose	More verbose output
–no-diffs	Do not output diffs of the refactoring
-w	Write back modified files
-n, –nobackups	Do not create backups of modified files
-o OUTPUT_DIR, –output-dir=OUTPUT_DIR	Place output files in this directory instead of overwriting input files. Requires the `-n` flag, as backup files are unnecessary when the input files are not modified.
-W, –write-unchanged-files	Write output files even is no changes were required. Useful with `-o` so that a complete source tree is translated and copied. Implies `-w` .
–add-suffix=ADD_SUFFIX	Specify a string to be appended to all output filenames. Requires `-n` if non-empty. Ex.: `–add-suffix='3'` will generate `.py3` files.

Remarks

The 2to3 tool is an python program which is used to convert the code written in Python 2.x to Python 3.x code. The tool reads Python 2.x source code and applies a series of fixers to transform it into valid Python 3.x code.

The 2to3 tool is available in the standard library as lib2to3 which contains a rich set of fixers that will handle almost all code. Since lib2to3 is a generic library, it is possible to write your own fixers for 2to3.