

```
1 1 ### #!/usr/bin/env python
2 1 ---
3 1 DOC """pycount.py -- A very initial effort to Python code metrics.
4 2 DOC
5 3 DOC This program started as a hack, bending and twisting pylint.py
6 4 DOC by Tim Peters. At that time I was interested on code metrics
7 5 DOC and thought let's do something quick and dirty. pycount then
8 6 DOC and today scans a Python file and reports back various cate-
9 7 DOC gories of lines it thinks it has found, like comments, doc
10 8 DOC strings, blank lines and, sometimes, real code as well.
11 9 DOC
12 10 DOC The output can be either as a table with only the number of
13 11 DOC lines found for each file or as a listing of the file(s)
14 12 DOC prefixed with some running numbers and classification for
15 13 DOC each line.
16 14 DOC
17 15 DOC The former is useful to scan a whole project e.g. when you
18 16 DOC need to know if the project is documented well or at all and
19 17 DOC where this info can be found. The latter is at least a nice
20 18 DOC new view to your own sources or that of others if nothing
21 19 DOC else!
22 20 DOC
23 21 DOC There are a couple of minor known bugs with pycount like:
24 22 DOC Doc strings must be tripple-quoted ones otherwise they are
25 23 DOC classified as normal source code. Continuation lines ending
26 24 DOC with a backslash are not treated at all. Complex regular ex-
27 25 DOC pressions (as in pycount itself) can knock the parser down,
28 26 DOC quickly. There is a built-in quick-and-dirty solution to
29 27 DOC this which might work whenever the problem is on one line
30 28 DOC only. But in "most cases" it works...
31 29 DOC
32 30 DOC Usage:
33 31 DOC
34 32 DOC pycount.py [-v] <file1> [<file2> ...]
35 33 DOC pycount.py [-v] <expr>
36 34 DOC pycount.py [-F] <linetypes> <file>
37 35 DOC pycount.py [-R] <expr>
38 36 DOC
39 37 DOC where <fileN> is a Python file (usually ending in .py),
40 38 DOC <linetypes> is a comma-seperated list of python line
41 39 DOC type codes (code, comment, doc string, blank)
42 40 DOC e.g. '###' or 'DOC,###,---'
43 41 DOC <expr> is a shell expression with meta-characters
44 42 DOC (note that for -R you must quote it)
45 43 DOC -v verbose flag, listing the source
46 44 DOC -F filter flag, listing the filtered source
47 45 DOC -R apply recursively on subdirectories
48 46 DOC
49 47 DOC NOTES:
50 48 DOC
51 49 DOC TODO:
52 50 DOC
53 51 DOC - Don't filter first line if '#!<path> python'(?).
54 52 DOC - De-obfuscate top-level if-stmt in main().
55 53 DOC - Improve usage as as a module.
56 54 DOC - Print statistics as percentage figures, maybe.
57 55 DOC - Write some test cases using pyunit.
58 56 DOC
59 57 DOC DONE:
60 58 DOC
61 59 DOC - Replace Unix 'find' with Python os.path.walk / fnmatch.
62 60 DOC - Scanning should also work recursively (-R option).
63 61 DOC - Test stdin case for single files.
64 62 DOC - Return total count per category when run on multiple files.
65 63 DOC - Add a feature to uncomment files.
66 64 DOC
67 65 DOC HISTORY:
68 66 DOC
69 67 DOC - 0.0.1 : 1997-??-?? : copy/past from Tim Peter's pylint
70 68 DOC - 0.0.2 : 1997-??-?? : included some refinements by Tim
```

```

71 69 DOC      - 0.0.3 : 1997-07-22 : doc & (C) (borrowed from M.-A. Lemburg)
72 70 DOC      - 0.0.4 : 1998-08-25 : replaced regex/regsub with re module,
73 71 DOC                added a global line counter in -v mode
74 72 DOC      - 0.0.5 : 1998-11-25 : code embellishments, recursive on files, ...
75 73 DOC      - 0.0.6 : 2000-07-04 : fixed typos, improved doc
76 74 DOC
77 75 DOC      FUTURE:
78 76 DOC
79 77 DOC      - The future is always uncertain...
80 78 DOC
81 79 DOC -----
82 80 DOC
83 81 DOC (c) Copyright by Dinu C. Gherman, 1998, gherman@europemail.com
84 82 DOC
85 83 DOC      Permission to use, copy, modify, and distribute this software and its
86 84 DOC      documentation without fee and for any purpose, except direct commercial
87 85 DOC      advantage, is hereby granted, provided that the above copyright notice
88 86 DOC      appear in all copies and that both that copyright notice and this
89 87 DOC      permission notice appear in supporting documentation.
90 88 DOC
91 89 DOC      THE AUTHOR DINU C. GHERMAN DISCLAIMS ALL WARRANTIES WITH REGARD TO
92 90 DOC      THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
93 91 DOC      FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL,
94 92 DOC      INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING
95 93 DOC      FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
96 94 DOC      NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION
97 95 DOC      WITH THE USE OR PERFORMANCE OF THIS SOFTWARE!
98 96 DOC
99 97 DOC      Dinu C. Gherman,
100 98 DOC
101 99 DOC      2000-07-04
102 100 DOC ""
103 2 ---
104 3 ---
105 1 COD __version__ = 0,0,6
106 4 ---
107 5 ---
108 2 COD import sys
109 3 COD import os
110 4 COD import re
111 5 COD import string
112 6 COD import getopt
113 7 COD import fnmatch
114 6 ---
115 7 ---
116 2 ### # Compile helper regular expressions.
117 8 ---
118 3 ### # Reg. exps. to find the end of a triple quote, given that
119 4 ### # we know we're in one; use the "match" method; .span()[1]
120 5 ### # will be the index of the character following the final
121 6 ### # quote.
122 8 COD _squote3_finder = re.compile(
123 9 COD     r"([\^'|]"
124 10 COD     r"\.|"
125 11 COD     r"'[\^\\']|"
126 12 COD     r"'\.|"
127 13 COD     r"'([\^\\']|"
128 14 COD     r"'\.)*''")
129 9 ---
130 15 COD _dquote3_finder = re.compile(
131 16 COD     r'([\^\\"]|'
132 17 COD     r'\.|"
133 18 COD     r'"[\^\\"]|'
134 19 COD     r'"\.|"
135 20 COD     r'"([\^\\"]|'
136 21 COD     r'"\.)*""')
137 10 ---
138 7 ### # Reg. exps. to find the leftmost one-quoted string; use the
139 8 ### # "search" method; .span()[0] bounds the string found.
140 22 COD _dquotel1_finder = re.compile(r'([\^"]|\.)*''')

```

```

141 23 COD _squotel_finder = re.compile(r"'([\^']|\.)+'"')
142 11 ---
143 9 ### # _is_comment matches pure comment line.
144 24 COD _is_comment = re.compile(r"^[ \t]*#").match
145 12 ---
146 10 ### # _is_blank matches empty line.
147 25 COD _is_blank = re.compile(r"^[ \t]*$").match
148 13 ---
149 11 ### # find leftmost splat or quote.
150 26 COD _has_nightmare = re.compile(r"""\["#]""").search
151 14 ---
152 12 ### # _is_doc_candidate matches lines that start with a triple quote.
153 27 COD _is_doc_candidate = re.compile(r"^[ \t]*(''|\"\"|\")")
154 15 ---
155 28 COD del re
156 16 ---
157 29 COD doc_type, comment_type, cod_type, blank_type = 'DOC', '###', 'COD', '---'
158 17 ---
159 18 ---
160 30 COD class Formatter:
161 31 COD     "Sort of a formatter class for filtering Python source code."
162 19 ---
163 32 COD     def __init__(self, showLineNums=1):
164 33 COD         "Init."
165 20 ---
166 34 COD         self.showLineNums = showLineNums
167 35 COD         self.showLineType = 0
168 36 COD         self.showLine = 0
169 37 COD         self.filterLineTypes = []
170 21 ---
171 38 COD     def print_line(self, gnum, cnum, type, line):
172 39 COD         "Print a line, prefixed with some type and count information."
173 22 ---
174 40 COD         if self.filterLineTypes and type in self.filterLineTypes:
175 41 COD             return
176 23 ---
177 42 COD         if self.filterLineTypes:
178 43 COD             print '%s' % line,
179 44 COD             return
180 24 ---
181 45 COD         if self.showLineNums and self.showLineType:
182 46 COD             if self.showLine and not self.filterLineTypes:
183 47 COD                 print '%5d %5d %3s %s' % (gnum, cnum, type, line),
184 25 ---
185 26 ---
186 48 COD def crunch(getline, filename, mode):
187 49 COD     "Parse the given file."
188 27 ---
189 13 ###     # for speed, give local names to compiled reps
190 50 COD     is_blank, is_comment, has_nightmare, is_doc_candidate = \
191 51 COD         _is_blank, _is_comment, _has_nightmare, _is_doc_candidate
192 28 ---
193 52 COD     quote3_finder = { '":': _dquote3_finder,
194 53 COD                       "'": _squote3_finder }
195 54 COD     quotel_finder = { '":': _dquotel_finder,
196 55 COD                       "'": _squotel_finder }
197 29 ---
198 56 COD     from re import sub
199 30 ---
200 57 COD     num_code = num_comment = num_blank = num_doc = num_ignored = 0
201 58 COD     in_doc = in_triple_quote = lineno = 0
202 59 COD     while 1:
203 60 COD         # Eat one line.
204 61 COD         classified = 0
205 62 COD         lineno, line = lineno + 1, getline()
206 63 COD         if not line:
207 64 COD             break
208 31 ---
209 64 COD         if in_triple_quote:
210 65 COD             if in_doc:

```

```

211 66 COD          num_doc = num_doc + 1
212 67 COD          mode.print_line(lineno, num_doc, doc_type, line)
213 68 COD          else:
214 69 COD              num_code = num_code + 1
215 70 COD              mode.print_line(lineno, num_code, cod_type, line)
216 71 COD          classified = 1
217 72 COD          m = in_triple_quote.match(line)
218 73 COD          if m == None:
219 74 COD              continue
220 15 ###          # Get rid of everything through the end of the triple.
221 75 COD          end = m.span()[1]
222 76 COD          line = line[end:]
223 77 COD          in_doc = in_triple_quote = 0
224 32 ---
225 78 COD          if is_blank(line):
226 79 COD              if not classified:
227 80 COD                  num_blank = num_blank + 1
228 81 COD                  mode.print_line(lineno, num_blank, blank_type, line)
229 82 COD              continue
230 33 ---
231 83 COD          if is_comment(line):
232 84 COD              if not classified:
233 85 COD                  num_comment = num_comment + 1
234 86 COD                  mode.print_line(lineno, num_comment, comment_type, line)
235 87 COD              continue
236 34 ---
237 16 ###          # Now we have a code line, a doc start line, or crap left
238 17 ###          # over following the close of a multi-line triple quote; in
239 18 ###          # (& only in) the last case, classified==1.
240 88 COD          if not classified:
241 89 COD              if is_doc_candidate.match(line):
242 90 COD                  num_doc = num_doc + 1
243 91 COD                  in_doc = 1
244 92 COD                  mode.print_line(lineno, num_doc, doc_type, line)
245 93 COD              else:
246 94 COD                  num_code = num_code + 1
247 95 COD                  mode.print_line(lineno, num_code, cod_type, line)
248 35 ---
249 19 ###          # The only reason to continue parsing is to make sure the
250 20 ###          # start of a multi-line triple quote isn't missed.
251 96 COD          while 1:
252 97 COD              m = has_nightmare(line)
253 98 COD              if not m:
254 99 COD                  break
255 100 COD             else:
256 101 COD                 i = m.span()[0]
257 36 ---
258 102 COD             ch = line[i] # splat or quote
259 103 COD             if ch == '#':
260 21 ###                 # Chop off comment; and there are no quotes
261 22 ###                 # remaining because splat was leftmost.
262 104 COD                 break
263 23 ###             # A quote is leftmost.
264 105 COD             elif ch*3 == line[i:i+3]:
265 24 ###                 # at the start of a triple quote
266 106 COD                 in_triple_quote = quote3_finder[ch]
267 107 COD                 m = in_triple_quote.match(line, i+3)
268 108 COD                 if m:
269 25 ###                     # Remove the string & continue.
270 109 COD                     end = m.span()[1]
271 110 COD                     line = line[:i] + line[end:]
272 111 COD                     in_doc = in_triple_quote = 0
273 112 COD                 else:
274 26 ###                     # Triple quote doesn't end on this line.
275 113 COD                     break
276 114 COD             else:
277 27 ###                 # At a single quote; remove the string & continue.
278 115 COD                 prev_line = line[:i]
279 116 COD                 line = sub(quotel_finder[ch], ' ', line, 1)
280 28 ###                 # No more change detected, so be quiet or give up.

```

```

281 117 COD             if prev_line == line:
282 29  ###             # Let's be quiet and hope only one line is affected.
283 118 COD             line = ""
284 30  ###             # raise "ParseError", "Giving up at line %d" % lineno
285 37  ---
286 119 COD         answer = lineno-1, num_code, num_doc, num_comment, num_blank, filename
287 120 COD         linenoSum = num_code + num_doc + num_comment + num_blank + 1 - num_ignored
288 121 COD         if lineno != linenoSum:
289 122 COD             reason = ('internal inconsistency in counts', lineno, answer)
290 123 COD             raise SystemError, reason
291 38  ---
292 124 COD         return answer
293 39  ---
294 40  ---
295 125 COD def formatHeaderLine():
296 126 COD     format = '%8s%8s%8s%8s%8s %s'
297 127 COD     return format % ('lines', 'code', 'doc', 'comment', 'blank', 'file')
298 41  ---
299 42  ---
300 128 COD def formatResultLine(resTuple):
301 31  ###     # Print countings.
302 129 COD     format = '%8s%8s%8s%8s%8s %s'
303 130 COD     return format % resTuple
304 43  ---
305 44  ---
306 131 COD def collectFiles(listPatCwd, dirname, names):
307 132 COD     "Recursively add filenames matching a certain pattern to a list."
308 45  ---
309 133 COD     list, pat, cwd = listPatCwd
310 134 COD     l = len(cwd)
311 135 COD     for name in names:
312 136 COD         p = os.path.join(dirname, name)
313 137 COD         if os.path.isfile(p) and fnmatch.fnmatch(name, pat):
314 32  ###             # Strip-off the current working directory from
315 33  ###             # the full path and replace it with a '.'.
316 138 COD             list.append('.' + p[1:])
317 46  ---
318 47  ---
319 139 COD def main():
320 140 COD     allTuples = []
321 141 COD     all = [0, 0, 0, 0, 0]
322 48  ---
323 142 COD     verbose = 0
324 143 COD     recursive = 0
325 144 COD     is_filtered = 0
326 145 COD     is_first = 1
327 146 COD     m = Formatter()
328 49  ---
329 147 COD     opts, args = getopt.getopt(sys.argv[1:], "vRF:")
330 148 COD     for o, a in opts:
331 149 COD         if o == '-v':
332 150 COD             m.showLineNums = 1
333 151 COD             m.showLineType = 1
334 152 COD             m.showLine = 1
335 153 COD         elif o == '-R':
336 154 COD             recursive = 1
337 155 COD         elif o == '-F':
338 156 COD             is_filtered = 1
339 157 COD             if string.find(a, ","):
340 158 COD                 for t in string.split(a, ","):
341 159 COD                     m.filterLineTypes.append(t)
342 160 COD             else:
343 161 COD                 m.filterLineTypes.append(a)
344 50  ---
345 34  ###     # Handle stdin case.
346 162 COD     if len(args) == 0:
347 163 COD         resTuple = crunch(sys.stdin.readline, '<stdin>', m)
348 164 COD         if is_first and not is_filtered:
349 165 COD             print formatHeaderLine()
350 166 COD         if not is_filtered:

```

```
351 167 COD          print formatResultLine(resTuple)
352 168 COD          return
353 51 ---
354 35 ###          # Handle all other cases.
355 52 ---
356 36 ###          # Find files if we are in recursive mode.
357 169 COD          if recursive:
358 170 COD              pat, cwd = args[0], os.getcwd()
359 171 COD              args = []
360 172 COD              os.path.walk(cwd, collectFiles, (args, pat, cwd))
361 53 ---
362 173 COD          for path in args:
363 174 COD              try:
364 175 COD                  f = open(path, "r")
365 176 COD              except IOError, details:
366 177 COD                  print "couldn't open %s: %s" % (path, details)
367 178 COD              else:
368 179 COD                  try:
369 180 COD                      resTuple = crunch(f.readline, path, m)
370 181 COD                      allTuples.append(resTuple)
371 182 COD                      if is_first and not is_filtered:
372 183 COD                          print formatHeaderLine()
373 184 COD                      if not is_filtered:
374 185 COD                          print formatResultLine(resTuple)
375 186 COD                  finally:
376 187 COD                      is_first = 0
377 188 COD                      f.close()
378 54 ---
379 37 ###          # Print total number in case we have more than one file.
380 189 COD          if len(args) > 1:
381 190 COD              for t in allTuples:
382 191 COD                  for i in (0, 1, 2, 3, 4):
383 192 COD                      all[i] = all[i] + t[i]
384 55 ---
385 193 COD              all.append("total")
386 194 COD              print formatResultLine(tuple(all))
387 56 ---
388 57 ---
389 195 COD if __name__ == '__main__':
390 196 COD     main()
lines  code  doc comment  blank  file
  390   196   100      37      57  C:\Tmp\pycount.py
```