Python Codecount™ Counting Standard

University of Southern California

Center for Systems and Software Engineering

October, 2007

Revision Sheet

Date	Version	Revision Description	Author
10/28/2007	1.0	Consolidated Draft	CSSE
04/02/2008	1.1	Update section 3.0 (selection, iteration statement)	CSSE
04/14/2008	1.2	Update section 3.0 (jump, expression statement)	CSSE

1.0 CHECKLIST FOR SOURCE STATEMENT COUNTS

PHYSICAL AND LOGICAL SLOC COUNTING RULES

Measurement Unit	Order of Precedence	Physical SLOC	Logical SLOC	Comments
Executable lines	1	One per line	See table below	Defined in 2.9
Non-executable lines				
Declaration (Data) lines	2	One per line	See table below	Defined in 2.4
Compiler directives	3	One per line	See table below	Defined in 2.5
Comments				Defined in 2.8
On their own lines	4	Not included (NI)	NI	
Embedded	5	NI	NI	
Banners	6	NI	NI	
Empty comments	7	NI	NI	
Blank lines	8	NI	NI	Defined in 2.7

Table 1 Physical and Logical SLOC Counting Counts

LOGICAL SLOC COUNTING RULES

No.	Structure	Order of Precedence	Logical SLOC Rules	Comments
R01	"*for*", "foreach", "*while*" or "*if*" statement	1	Count once.	"*while*" is an independent statement.
R02	*do* {…} *until* (…); statement	2	Count once.	Braces *{…}* and semicolon *;* used with this statement are not counted.
R03	Statements ending by a semicolon	3	Count once per statement, including empty statement.	Semicolons within "*for*" statement are not counted. Semicolons used with R01 and R02 are not counted.
R04	Block delimiters, braces {…}	4	Count once per pair of braces *{..}*, except where a closing brace is followed by a semicolon, i.e. *};* *or an opening brace comes after a keyword "else"*	Braces used with R01 and R02 are not counted. Function definition is counted once since it is followed by *{…}.*
R05	Compiler directive	5	Count once per directive	

Table 2 Logical SLOC Counting Rules

2.0 DEFINITIONS

2.1 SLOC – Source Lines Of Code is a unit used to measure the size of SOFTWARE PROGRAM. SLOC counts the program source code based on a certain set of rules. SLOC is a key input for estimating project effort and is also used to calculate productivity and other measurements.

2.2 Physical SLOC – One physical SLOC is corresponding to one line starting with the first character and ending by a carriage return or an end-of-file marker of the same line, and which excludes the blank and comment line.

2.3 Logical SLOC – Lines of code intended to measure "statements", which normally terminate by a semicolon (C/C++, Java, C#) or a carriage return (VB, Assembly), etc. Logical SLOC are not sensitive to format and style conventions, but they are language-dependent.

2.4 Data declaration line or data line – A line that contains declaration of data and used by an assembler or compiler to interpret other elements of the program.

The following are the Python keywords that denote data declaration lines:

`class`			

Table 3 Data Declaration Types

NOTE: See Section 3 of this document for examples of data declaration lines.

2.5 Compiler directive - A statement that tells the compiler how to compile a program, but not what to compile.

2.6 Blank line – A physical line of code, which contains any number of white space characters (spaces, tabs, form feed, carriage return, line feed, or their derivatives).

2.7 Comment line – A comment is defined as a string of zero or more characters

There are two styles of comments in python

1) # single line comment
2) " " " this is a multiline comment
   which spawns many lines
   " " "

2.8 Executable line of code - A line that contains software instruction executed during runtime and on which a breakpoint can be set in a debugging tool. An instruction can be stated in a simple or compound form.

- An executable line of code may contain the following program control statements:
  - Selection statements (if, *?* operator)
  - Iteration statements (for, while, do-until, foreach)
  - Empty statements (pass)
  - Jump statements (return, goto, last, next, exit function)

- Expression statements (function calls, assignment statements, operations, etc.)
- Block statements

NOTE: See Section 3 of this document for examples of control statements.

- An executable line of code may not contain the following statements:
  - Whole line comments, including empty comments and banners
  - Blank lines

3.0 EXAMPLES OF LOGICAL SLOC COUNTING

Executable lines

SELECTION STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
	if, else if, else and nested if statements	if <expression>: <statements>	if password == "pass": print "Access Granted"	1 1

		if <expression>:
		  <statement>
		else:
		  <statement>
			if password == "name":
			  print "Access Granted"
			else:
			  print "Access Denied"
				1
				1
				0
				1

		if <expression>:
		  <statements>
		elif <expression>:
		  <statements>
		else:
		  <statements>
			if num > 0:
			  print 'positive'
			elif num < 0:
			  print 'negative'
			else:
			  print 'zero'
				1
				1
				1
				1
				0
				1

		if <expression>:
		  <statements>
		  <statements>
		else:
		  <statements>
		
		NOTE: complexity is not considered
			if x < 0:
			  x = 0
			  print 'Negative'
			else:
			  print 'Positive'
				1
				1
				1
				0
				1

	try-except-finally	try:
		  <do something>
		except Exception:
		  <handle the error>
		finally:
		  <cleanup>
			try:
			  try: 1/0
			  except:
			  print "exception"
			except ZeroError:
			  print "divide-by-0"
				1
				1
				1
				1
				1
				1

ITERATIONS STATEMENTS

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT
	for	for <expression>: <*statement*>	for x in a: print x,	1 1

		NOTE: "for" statement counts as one, no matter how many optional expressions it contains	for x in a:
			{
			  print 'x'
			}
				1
				0
				1
				0

	while	While <boolean expression>:
		  <statement>
			while x <= 100:
			  print x
			  x += 1
				1
				1
				1

JUMP STATEMENTS
(ARE COUNTED AS THEY INVOKE ACTION – PASS TO THE NEXT STATEMENT)

ID	STATEMENT DESCRIPTION	GENERAL FORM	SPECIFIC EXAMPLE	SLOC COUNT

	return	return *expression*	def knights():
			  title = 'Sir'
			  action = (lambda x: title + ' ' + x)
			  return action
			act = knights()
			print act('robin')
				1
				1
				1
				1
				1
				1

	break	break	for x in range(1, 11):
			  if x == 5:
			  break
			  print x,
			print "\nBroke out of loop at x =", x
				1
				1
				1
				1
				1

	exit function	os.exit(int return_code)	def outahere():
			  import os
			  print 'Bye os world'
			  os._exit(99)
			  print 'Never reached'
			
			if __name__ == '__main__': outahere()
				1
				1
				1
				1
				1
				-
				2

	continue	continue	for x in range(1, 11):
			  if x == 5:
			  continue
			  print x,
			print "\nUsed continue to skip printing the value 5"
				1
				1
				1
				1
				1

EXPRESSION STATEMENTS				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**
EES1	function call	<function_name> (<parameters>);	read_file (name);	1
EES2	assignment statement	assignment_stmt = (target_list "=")+ expression_list target_list = target ("," target)* [","] target = identifier | "(" target_list ")" | "[" target_list "]" | attributeref | subscription | slicing	x = y name = “file1” a, b, c = 1,2,3	1 1 1

EES3	empty statement (is counted as it is considered to be a placeholder for something to call attention)	Pass	if month == 1:
			  pass
			else:
			  print "late"
				1
				1
				1
				1

EES4	Explicit line joining	<expression> \
		<expression> \
		<expression>
			bar = 'this is ' \
			  'one long string ' \
			  'that is split ' \
			  'across multiple lines'
			print bar
				1
				-
				-
				-
				1

EES5	Implicit line joining (Expressions in parentheses, square brackets or curly braces can be split over more than one physical line without using backslashes.)	(<expression>,
		…
		<expression>)
		
		[<expression>,
		…
		<expression>]
		
		{<expression>,
		…
		<expression>}
			day = [‘mon’, ‘tue’,
			  ‘wed’, ‘thur’, ‘fri’,
			  ‘sat’, ‘sun’]
			
			def node(name):
			return {
			  'Parent' : None,
			  'LeftChild' : None,
			  'RightChild' : None,
			  'LeftRoutingTable' : list(),
			  'Name' : name,
			  'Level' : 0
			}
				1
				-
				-
				
				1
				1
				-
				-
				-
				-
				-
				-
				-

DECLARATION (DATA) LINES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**

DDL1	class	class ClassName:
		<statement-1>
		.
		.
		.
		<statement-N>
			class MyClass:
			i = 12345
			def f(self):
			return 'hello'
				0
				1
				1
				1

COMPILER DIRECTIVES				
ID	**STATEMENT DESCRIPTION**	**GENERAL FORM**	**SPECIFIC EXAMPLE**	**SLOC COUNT**