My first python program: can you tell me what I'm doing wrong?

Asked 15 years, 8 months ago Modified 14 years, 4 months ago

Viewed 913 times

6

I hope this question is considered appropriate for stackoverflow. If not, I'll remove the question right away.

I've just wrote my very first python program. The idea is that you can issue a command, and it's gets sent to several servers in parallel.

This is just for personal educational purposes. The program works! I really want to get better at python and therefore I'd like to ask the following questions:

1. My style looks messy compared to PHP (what I'm used to). Do you have any suggestions around style improvements.
2. Am I using the correct libraries? Am I using them correctly?
3. Am I using the correct datatypes? Am I using them correctly?

I have a good programming background, but it took me quite a while to develope a decent style for PHP (PEAR-coding standards, knowing what tools to use and when).

The source (one file, 92 lines of code)

http://code.google.com/p/floep/source/browse/trunk/floep

python

Share

Improve this question

Follow

edited Jun 26, 2009 at 4:52

Rex M

144k ● 34 ● 291 ● 315

asked Apr 4, 2009 at 1:27

Evert

99.2k ● 18 ● 130 ● 212

Your comments have all been very helpful! Thank you so much.. – Evert Apr 4, 2009 at 16:55

4 Answers

Sorted by: Highest score (default)

10

Usually is preferred that what follows after the end of sentence `:` is in a separate line (also don't add a space before it)

```
if options.verbose:
  print ""
```

instead of

```
if options.verbose : print ""
```

You don't need to check the len of a list if you are going to iterate over it

```
if len(threadlist) > 0 :
  for server in threadlist :
    ...
```

is redundant, a more 'readable' is (python is smart enough to not iterate over an empty list):

```
for server in threadlist:
  ...
```

Also a more 'pythonistic' is to use list's comprehensions (but certainly is a debatable opinion)

```
server = []
for i in grouplist : servers+=getServers(i)
```

can be shortened to

```
server = [getServers(i) for i in grouplist]
```

Share Improve this answer edited Aug 6, 2010 at 20:01

Follow

community wiki

3 revs
Ismael

I fixed all my if statements, but I have some issues with the other two.. About the while statement, it actually has a purpose here. The while statement is there to go through the loop until it's completely empty. I didn't really understand the section about list comprehension at all.. Upvoted! – Evert Apr 4, 2009 at 2:09

My mistake I didn't understand your code around the while. List comprehension are a very pythonistic way of writing code I can suggest to check python.org/dev/peps/pep-0202, or search for those terms in the net, there're nice tutorials about it. – Ismael Apr 4, 2009 at 3:33

2 re: "debatable opinion"... in this case, I think a list comprehension is entirely appropriate. Once you understand list comprehensions, there's no reason not to use one in code like this. – David Z Apr 4, 2009 at 3:49

7

Before unloading any criticism, first let me say congratulations on getting your first Python program working. Moving from one language to another can be a chore, constantly fumbling around with syntax issues and hunting through unfamiliar libraries.

The most quoted style guideline is PEP-8, but that's only a guide, and at least some part of it is ignored...no, I mean deemed not applicable to some specific situation with all due respect to the guideline authors and contributors :-).

I can't compare it to PHP, but compared to other Python applications it is pretty clear that you are following style conventions from other languages. I didn't always agree with many things that other developers said you **must** do, but over time I recognized why using conventions helps communicate what the application is doing and will help other developers help you.

Raise exceptions, not strings.

```
raise 'Server or group ' + sectionname + ' not
found in ' + configfile
```

becomes

```
raise RuntimeError('Server or group ' +
sectionname + ' not found in ' + configfile)
```

No space before the ':' at the end of an 'if' or 'for', and don't put multiple statements on the same line, and be consistent about putting spaces around operators. Use variable names for objects and stick with `i` and `j` for loop index variables (like our masterful FORTRAN forefathers):

```
for i in grouplist : servers+=getServers(i)
```

becomes:

```
for section in grouplist:
    servers += getServers(section)
```

Containers can be tested for contents without getting their length:

```
while len(threadlist) > 0 :
```

becomes

```
while threadlist:
```

and

```
if command.strip() == "" :
```

becomes

```
if command.strip():
```

Splitting a tuple is usually not put in parenthesis on the left hand side of a statement, and the command logic is a bit convoluted. If there are no args then the " ".join(...) is going to be an empty string:

```
(options,args) = parser.parse_args()

if options.verbose : print "floep 0.1"
```

```
command = " ".join(args)

if command.strip() == "" : parser.error('no
command given')
```

becomes

```
options, args = parser.parse_args()
if options.verbose:
    print "floep 0.1"

if not args:
    parser.error('no command given')
command = " ".join(args)
```

A python for loop has an unusual 'else' clause which is executed if the loop goes through all of the elements without a 'break':

```
    for server in threadlist :
        foundOne = False
        if not server.isAlive() :
            ...snip...
            foundOne = True

        if not foundOne :
            time.sleep(0.010)
```

becomes

```
    for server in threadlist:
        if not server.isAlive():
            ...snip...
            break
```

```
    else:
        time.sleep(0.010)
```

Getting a list of lines and then joining them back together is a bit long winded:

```
        result = proc.readlines()
        strresult = ''
        for line in result : strresult+=line
        self.result = strresult
```

becomes

```
        self.result = proc.read()
```

Your library use is good, check out the subprocess module, it's a little more up-to-date.

Your datatypes are fine.

And you'll get lots of other anwsers :-)

Share Improve this answer

Follow

answered Apr 4, 2009 at 2:37

Joel Bender

885 ● 4 ● 8

Very helpful information.. Thank you sir, I'll apply all your suggestions. – Evert Apr 4, 2009 at 16:56

5

String exceptions are deprecated in Python, so this line:

```
if not config.has_section(sectionname):
    raise 'Server or group ' + sectionname + ' not fou
```

should be reworked into something like this:

```
if not config.has_section(sectionname):
    raise ConfigNotFoundError(
        "Server or group" + sectionname + "not found i

class ConfigNotFoundError(Exception):
    pass
```

[Edited to reflect the suggestion of dangph in the comments]

It's more lines of code, but it's better for future upgrades.

For readability's sake, something like this:

```
parser.add_option('-q','--quiet',action="store_false",
server output", dest="verbose", default=True)
```

Can be rewritten like this:

```
parser.add_option('-q',
                  '--quiet',
                  action="store_false",
                  help="Display only server output",
                  dest="verbose",
                  default=True)
```

You might prefer another method of splitting the method call up, but the idea is that long lines can be hard to read.

You should also read PEP 8 to get a sense of Python style.

Share Improve this answer Follow

edited Apr 4, 2009 at 3:02

answered Apr 4, 2009 at 2:22

chradcliffe

471 ● 2 ● 10

To quickly define an exception type, you can just do this: "class ConfigNotFoundError(Exception): pass" (with "pass" on the next line of course). – dan-gph Apr 4, 2009 at 2:32

Great tip, and I also like the easier class definition from dangph – Evert Apr 4, 2009 at 16:57

3

Often, for reuse purposes, we do the following, starting at about line 48 in your program

```
def main():
    config = ConfigParser.RawConfigParser()
    etc.

if __name__ == "__main__":
    main()
```

This is just a starting point.

Once you've done this, you realize that main() is really two parts: parsing the command-line interface and doing the work. You then want to refactor things to look like this.

```
def serverWork(group,...):
    servers = getServers(group)
    etc.

def main():
    config = ConfigParser.RawConfigParser()

    if command.strip() == "":
        parser.error('no command given')
    else:
        serverWork( options.group, options.etc., ... )
```

Now, you have elevated the real work to a function within this module. Your serverWork function can now be reused easily by other programs or scripts.

Share Improve this answer

Follow

answered Apr 4, 2009 at 11:58

S.Lott

391k ● 82 ● 517 ● 788