WELCOME TO CLASS 4!

BLACK HAT PYTHON3

RALEIGH ISSA

GITHUB REPO

https://github.com/tiarno/bhp3_class

SUMMARY FROM LAST CLASS

- Joining threads (blocking)
- Briefly, context managers (`with ...`)
- lxml for web scraping
- sockets
- IP, ICMP headers and parsing

UDP SCANNER

- how it works
- UDP packet to unused port
  - network unreachable (from router)
  - host unreachable (from router)
  - port unreachable ! type 3, code 3

TEST IT OUT

- `scanner.py`

PYTHON DETAILS

- git
- imports
- ipaddress
- bytes/strings
- context managers

GIT LOCAL

Plain Git:

- add
- commit
- status
- git log --pretty=oneline

GIT AND GITHUB

- git fork is a GitHub thing only
- `git clone` (get a copy of a repo)
- git and your local repo:
  - add/commit changes
  - can have a remote (e.g., GitHub)
  - git pull (pulls down updates from the remote)
  - git push (pushes up changes to the remote of your repo)
  - can have an upstream (from the original fork: a GitHub thing)

IMPORTS

- `__init__.py`
- import os -> os.path, os.listdir()
- from lxml import etree -> etree.parse()
- import multiprocessing as mp -> mp.Process()
- from ctypes import * -> Structure

WORD FINDER FUNCTION

CODE REUSE!

- `bhp3_class/web/__init__.py` -> getwords()
- `dirfinder2.py`
- `wp_killer2.py`

ipaddress PACKAGE

```
IPv4Network('192.168.1.69/16')
  .hosts() -> iterator over usable hosts in network

ipaddress.ip_address(self.src)
```

NETWORK SCANNER CODE:

scanner.py

```
for ip in ipaddress.ip_network(SUBNET).hosts():
    send packet to ip
```

BYTES VS STRINGS

- sockets, processes return bytes
- bytes are the computer's language
- strings are our language
- string.encode() -> bytes
- bytes.decode() -> string

Usually the decoding is to a UTF-8 string

```
S.encode(encoding='utf-8', errors='strict') -> bytes
    Encode S using the codec registered for encoding.

B.decode(encoding='utf-8', errors='strict') -> string
    Decode the bytes using the codec registered for encoding.
```

Default encoding is 'utf-8'

CONTEXT MANAGERS

- As a class, define `__enter__` and `__exit__`
- As a generator (`try: finally:`)
  - code before the `yield` == `__enter__`
  - code in the `finally` block is the `__exit__`

```
@contextmanager
def some_generator(<arguments>):
    <setup>
    try:
        yield <value>
    finally:
        <cleanup>
```

So that

```
with some_generator(<arguments>) as <variable>:
   <body>
```

is equivalent to this:

```
<setup>
try:
   <variable> = <value>
   <body>
finally:
    <cleanup>
```

- Can use `contextlib.contextmanager` decorator
- Built in context managers (e. g., files, sockets)
- closing, redirect_stdout, see doc for more…

```
with closing(<module>.open(<arguments>)) as f:
    <block>
```

is equivalent to this:

```
f = <module>.open(<arguments>)
try:
    <block>
finally:
    f.close()
```

redirect_stdout

How to write help() to a file

```
with open('help.txt', 'w') as f:
    with redirect_stdout(f):
        help(pow)
```

YOUR JOB

- Write your own network scanner, place it in `packets` module
- Read about context managers
- Become familiar with bytes vs strings, encoding vs decoding.

FEEDBACK PLEASE!

- tim@reachtim.com
- discord: https://discord.gg/WR23qUj

EXTRA LINKS

- https://www.securitywizardry.com/index.php/tools/p
  headers.html
- https://stackoverflow.com/questions/9257533/what-i
  difference-between-origin-and-upstream-on-
  github/9257901#9257901
- http://ndpsoftware.com/git-
  cheatsheet.html#loc=remote_repo;