Python “psutil” module

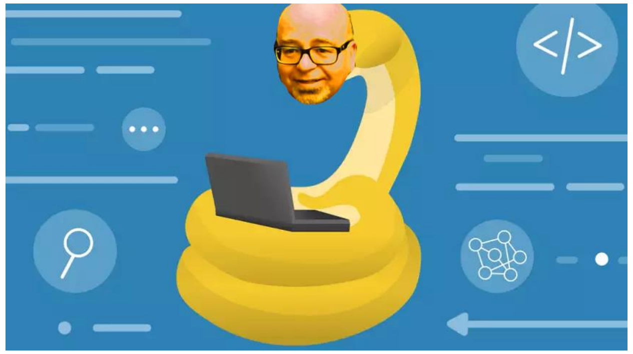
Exploring in Windows

Author: Joff Thyer © February 2024

About Joff

- Black Hills Information Security
- AntiSyphon Instructor
- Research/Dev/Initial Access ops.
- Unofficial title of “Malware Pit Boss”

- Credit to @OrthicOn for creating the Joff/Yoda meme.



Python "psutil" module!

- Python "pypi" package (use pip to install)
  - Note: Ubuntu like systems prefer: "apt install python3-psutil"
- Cross platform module for process and system information
  - Linux
  - Windows
  - macOS
  - FreeBSD / OpenBSD / NetBSD
- Python version 2.7 and 3.6+

Scope of coverage

- Implements most functionalities covered by UNIX like tools
  - ps, top, iotop
  - lsof
  - ifconfig
  - netstat
- BSD 3-Clause License
- Authored by Giampaolo Rodola

Why am I talking about this?

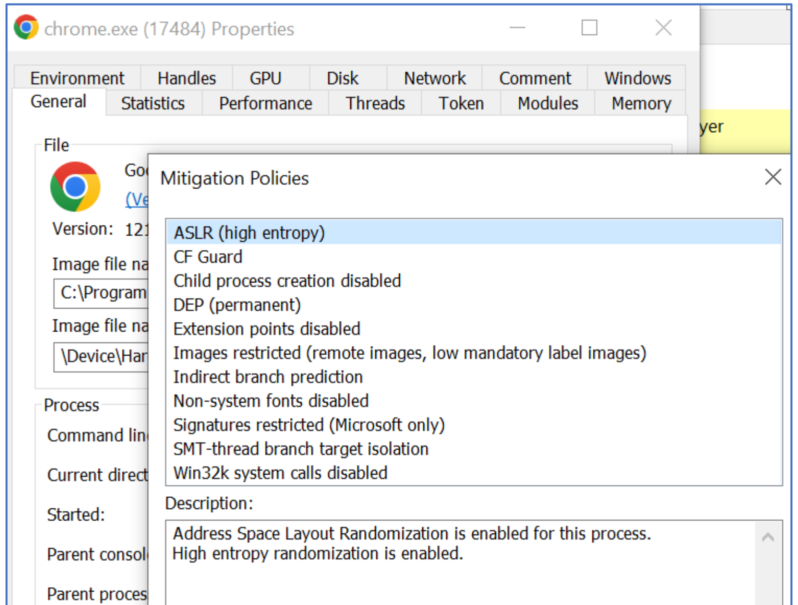
- I often do malware dev work under Windows
- Examining process details during malware R&D is helpful
- Great GUI tools like "System Informer" are helpful
  - Formally named "process hacker"
- I like the "speed" of command line tools
- I am a Windows Terminal and PowerShell 7.x fan these days.
- I was "researching" Chromium process architecture…

Chrome/Chromium processes

- Chromium uses an InterProcess Communications (IPC) abstraction library called "MojoM".
- Processes have various dedicated tasks
- These dedicated options are specified on the command line
  - `"–type" and "–utility-sub-type" flag`
  - Type:
    - Renderer, Utility, GPU-Process, Crashpad-Handler
  - Utility-Sub-Type
    - Storage.mojom.StorageService
    - Network.mojom.NetworkService

Chromium continued..

- Since about 2021, some (not all) of the Chromium processes are protected under Windows
  - This applies to both the Chrome and MSEdge browser
  - Mitigations via the Windows ProcessThreads API.
- Multiple mitigations are applied to types "renderer" and "gpu-process" processes



Version: 12

Image file na C:\Program

Image file na \Device\Har

ASLR (high entropy)
CF Guard
Child process creation disabled
DEP (permanent)
Extension points disabled
Images restricted (remote images, low mandatory label images)
Indirect branch prediction
Non-system fonts disabled
Signatures restricted (Microsoft only)
SMT-thread branch target isolation
Win32k system calls disabled

Description:
Address Space Layout Randomization is enabled for this process.
High entropy randomization is enabled.

Some Chrome Renderer process mitigations

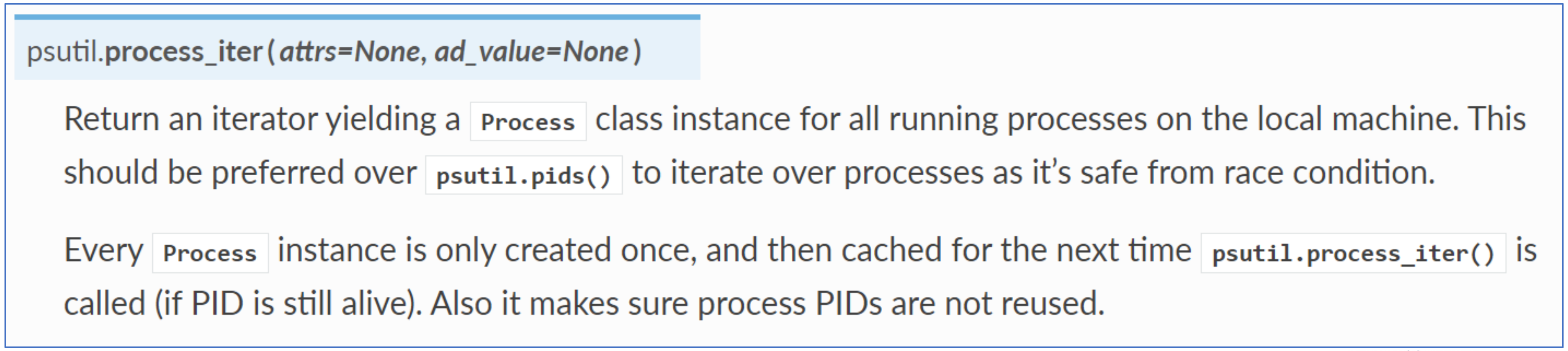
- Control Flow Guard
- Child process creation disabled
- Images restricted
  - Restricts the type of executable images that can be mapped into the process. (no low-mandatory label or remote devices)
- Indirect branch prediction
  - Requires CPU hardware support (Intel)
- Signatures Restricted (Microsoft only)
  - Only allow Microsoft signed images to be loaded into process
- Win32k system calls disabled
  - Disable User/GDI function related system calls (gpu process will do this..)

Why does this matter?

- Hacking chrome/chromium gives us access to interesting data
- The renderer type processes are often attacked via HTML/Typescript/CSS content
- We want to mitigate attacks from other O/S processes (malware)
  - Attacking a renderer or GPU process from the O/S is very difficult.
  - Attacking the network service process is easier.

Psutil process_iter()

- Returns a Python iterable object yielding a "Process" class instance
- Process() class instances are created once then cached
- Can use the "attrs" parameter to retrieve specific attributes
  - Defaults to ALL which is slow



psutil.**process_iter**(*attrs=None, ad_value=None*)

Return an iterator yielding a `Process` class instance for all running processes on the local machine. This should be preferred over `psutil.pids()` to iterate over processes as it's safe from race condition.

Every `Process` instance is only created once, and then cached for the next time `psutil.process_iter()` is called (if PID is still alive). Also it makes sure process PIDs are not reused.

What is a Python “iterable” object?

- An object that can be iterated over
- An object we can use within a Python “loop”
- An object that implements the Python iterable protocol
    - Implements a method called “__iter__()” or “__getitem__()”
    - The iter method returns an object containing a “__next__()” method
    - “__next__()” retrieves the “next” item!

process_iter() “attrs” argument

- “attrs” are the available Process() class attribute names
- The more attributes you ask for, the longer it will take… ☺
- Some useful “attrs” are:
  - name
  - pid
  - username
    - Process owner
  - cmdline
    - the actual command line arguments
  - environ
  - memory_info
    - returns a named tuple of fields containing memory information
  - num_threads

memory_info()

- All values expressed in bytes. (rss and vms common across all platforms)
- Windows has many extra parameters returned.
  - rss (alias for wset)
  - vms (alias for pagefile)
  - num_page_faults
  - peak_wset / wset
  - peak_paged_pool / paged_pool
  - peak_nonpaged_pool / nonpaged_pool
  - peak_pagefile / pagefile
  - private

memory_maps()

- Returns a list of named tuples with process memory region information.
- Under Windows, returns only two items:
  - Loaded module name (DLL)
  - Memory size in bytes of loaded module (rss)

What is really happening on Windows?

- The "C" code that implement "psutil" is using:
  - the Windows process status API for memory_info() and memory_maps().
    - https://learn.microsoft.com/en-us/windows/win32/psapi/process-status-helper
  - And the tool help library for process state snapshots
    - CreateToolhelp32Snapshot() ☺
    - https://learn.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot

Possible Challenges

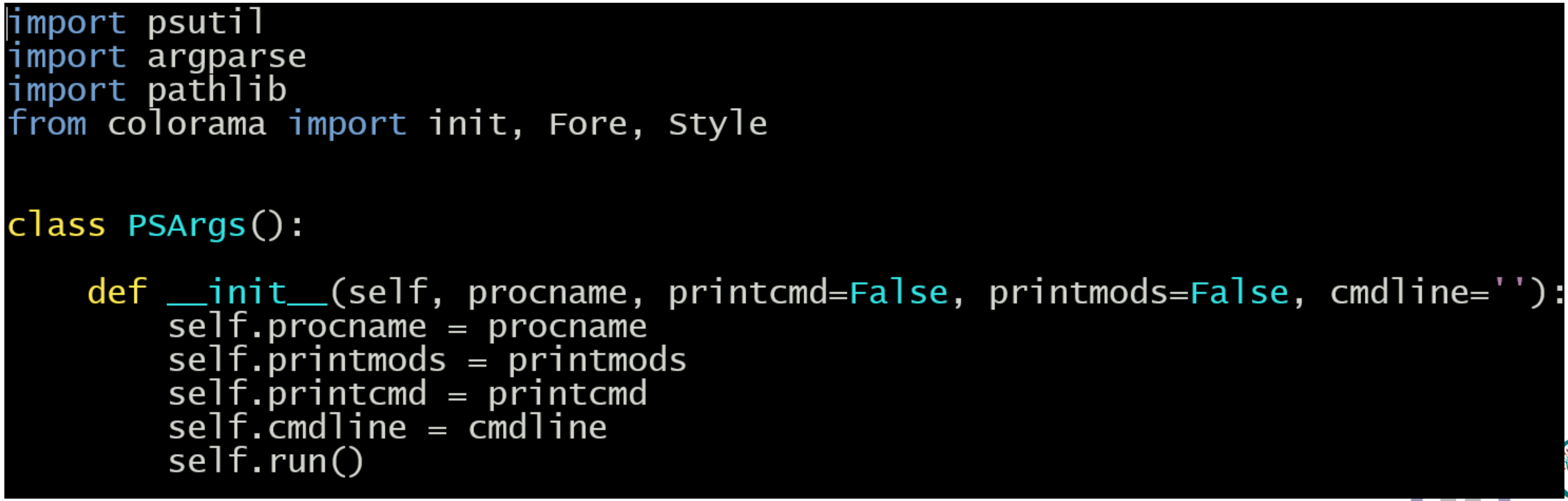
- The “psutil” Process class will raise an exception under certain conditions
- Commonly in Windows, these are:
  - psutil.NoSuchProcess(): when a process no longer exists
  - psutil.AccessDenied(): insufficient privileges to perform an action.
    - Commonly raised when attempting to use OpenProcess()
  - psutil.TimeoutExpired()
    - raised when waiting for a process to terminate within a specific timeout period, and that period elapses.
- For “process_iter()” and related memory_info(), “AccessDenied” is most common.

Back to my goals and preferences

- I like command line tools
- I was working with Chrome research
- I was interested in process command line arguments, and loaded modules
- I wanted a clear unambiguous process listing readout
- I wanted to sort loaded modules by memory footprint and optionally display
- I wanted some filtering options

Structure of my “ps.py” script

- Utilizes additional modules
  - pathlib, argparse, and colorama
  - “colorama” gives us nice pretty colors for display
  - 100% implemented in a Python class called “PSArgs()”



```
import psutil
import argparse
import pathlib
from colorama import init, Fore, Style


class PSArgs():

    def __init__(self, procname, printcmd=False, printmods=False, cmdline=''):
        self.procname = procname
        self.printmods = printmods
        self.printcmd = printcmd
        self.cmdline = cmdline
        self.run()
```

Python3 language elements we are using

- Imported Modules
  - Builtin modules "pathlib", and "argparse"
  - 3rd party modules "psutil", and "colorama"
- Classes, attributes and methods
- Format strings (f-strings), and printed output
- For loops and conditional logic with "continue" statement
- Dictionaries and lists
- Exception Handling
- String methods such as "lower()"
- The sorted() method and lambda functions

Python classes

- Python is a fully object-oriented language
- Everything in Python is an object!
- A Python class is a “blueprint” for creating an object
- A Python object is an instance of a class.
- Objects have attributes and methods associated with them
  - A “method” is simply a function encapsulated within an object
  - An “attribute” is simply a variable encapsulated within an object

Python dictionaries and lists

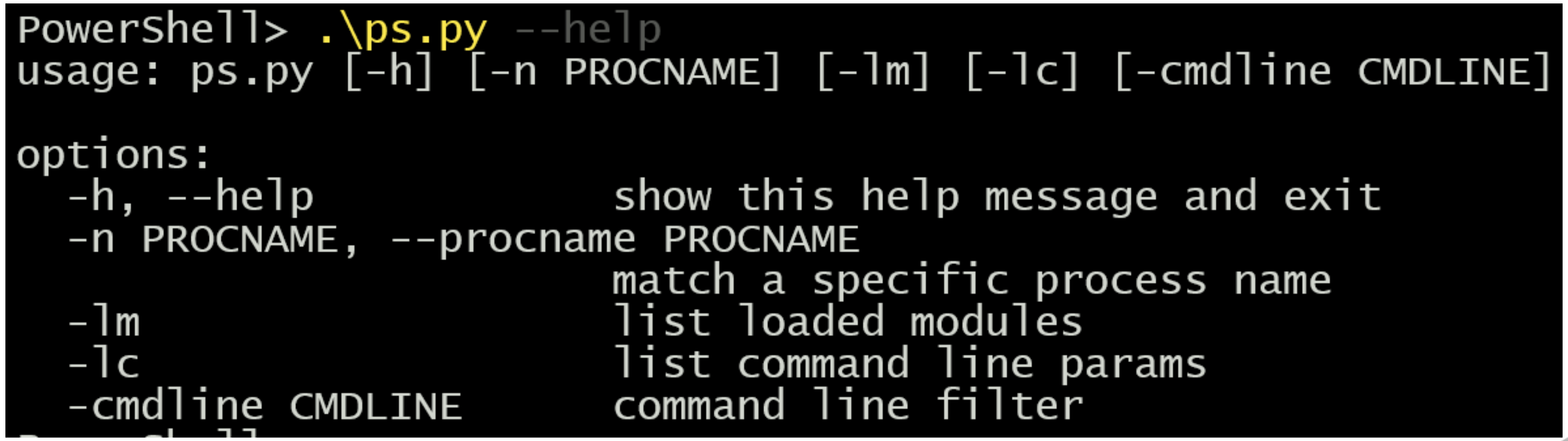
- A Python dictionary is a data structure also known as a hash table
  - Stores items in key/value pairs.
    - The key can be almost any object, but very commonly a string like “joff”
    - The value can also be any Python object, like a string, list or tuple
  - Since the keys are hashed, looking up data is very fast/efficient
- A list is a data structure that stores elements by an integer index.
  - A list can contain almost any Python object also
  - A list can be sorted using the “sorted()” method.
    - Note: values must be comparable and same object types to sort!

Python Exception Handling

- Simply provides us a way to catch unusual errors
- If we don’t use any exception handling, an unusual error would force a script to crash
- There are named exceptions which are documented
- We can catch the errors either by name or generically.

Command line arguments

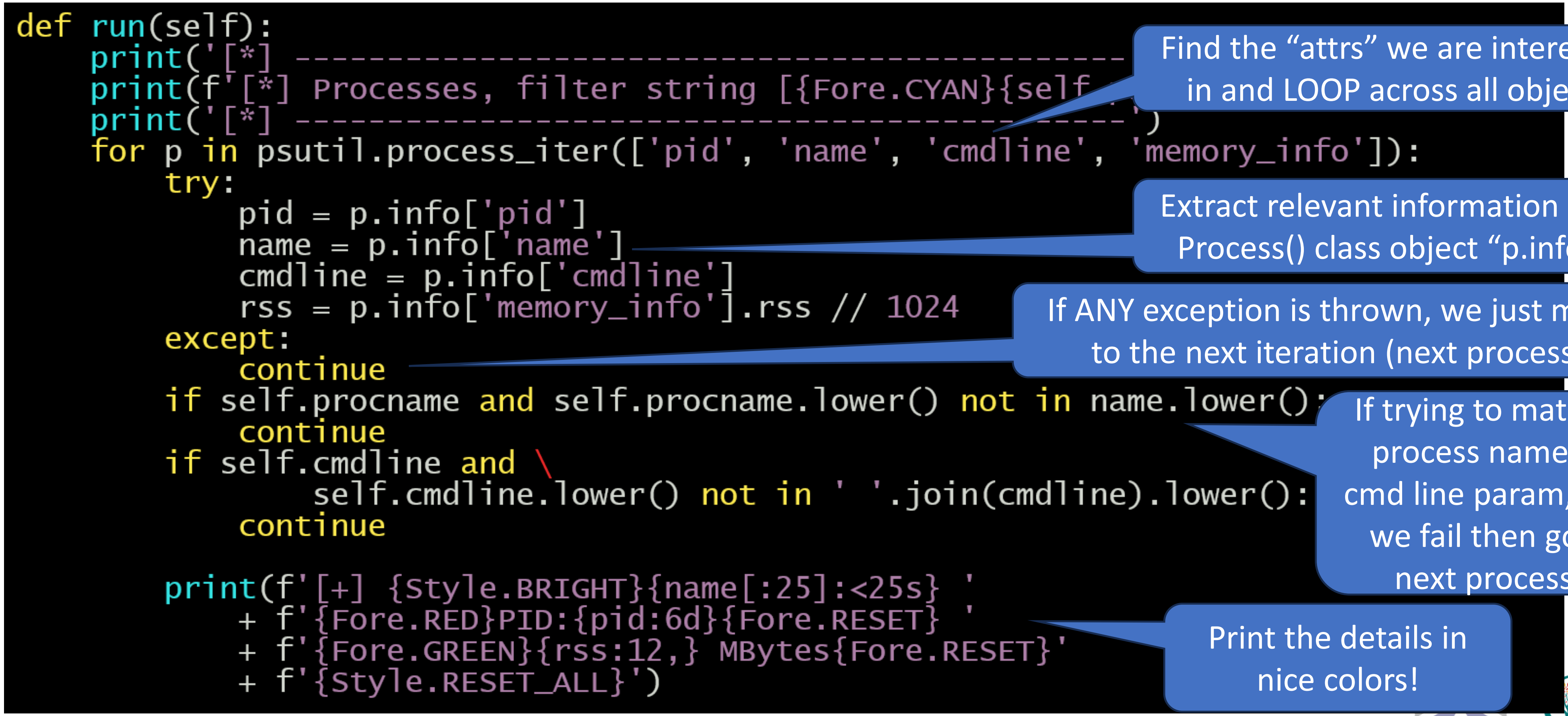
- Using "argparse" module, I implemented
  - A process name matching parameter
  - A parameter to list loaded modules
  - A parameter to list command line information
  - A parameter to filter on command line information



```
PowerShell> .\ps.py --help
usage: ps.py [-h] [-n PROCNAME] [-lm] [-lc] [-cmdline CMDLINE]

options:
  -h, --help            show this help message and exit
  -n PROCNAME, --procname PROCNAME
                        match a specific process name
  -lm                   list loaded modules
  -lc                   list command line params
  -cmdline CMDLINE      command line filter
```

The core of the script!



```
def run(self):
    print('[*] ----------------------------------------------
    print(f'[*] Processes, filter string [{Fore.CYAN}{self
    print('[*] ----------------------------------------------')
    for p in psutil.process_iter(['pid', 'name', 'cmdline', 'memory_info']):
        try:
            pid = p.info['pid']
            name = p.info['name']
            cmdline = p.info['cmdline']
            rss = p.info['memory_info'].rss // 1024
        except:
            continue
        if self.procname and self.procname.lower() not in name.lower():
            continue
        if self.cmdline and \
                self.cmdline.lower() not in ' '.join(cmdline).lower():
            continue

        print(f'[+] {Style.BRIGHT}{name[:25]:<25s} '
            + f'{Fore.RED}PID:{pid:6d}{Fore.RESET} '
            + f'{Fore.GREEN}{rss:12,} MBytes{Fore.RESET}'
            + f'{Style.RESET_ALL}')
```

Find the “attrs” we are interested in and LOOP across all objects

Extract relevant information from Process() class object “p.info()”

If ANY exception is thrown, we just move to the next iteration (next process)

If trying to match a process name or cmd line param, and we fail then goto next process!

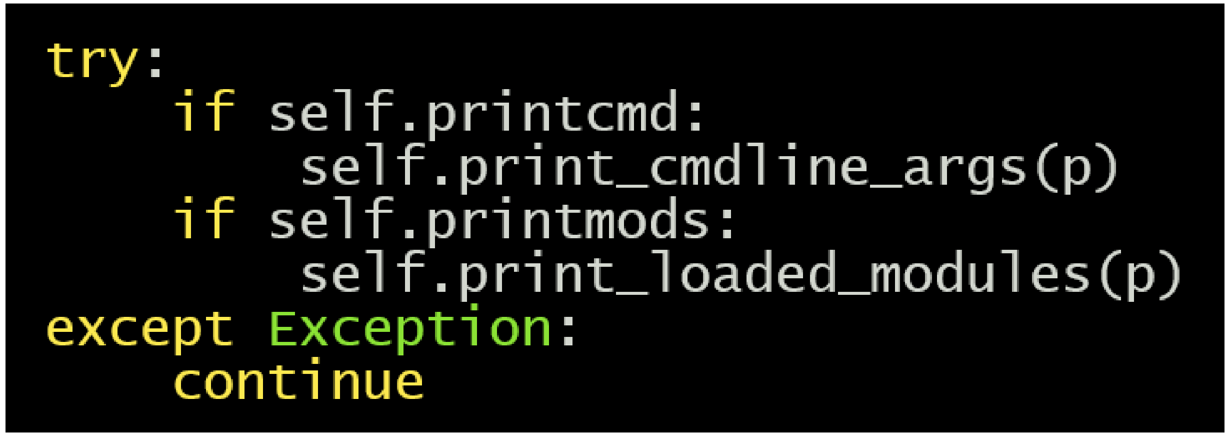
Print the details in nice colors!

RIVER GUM SECURITY

BLACK HILLS Information Security

Printing additional info

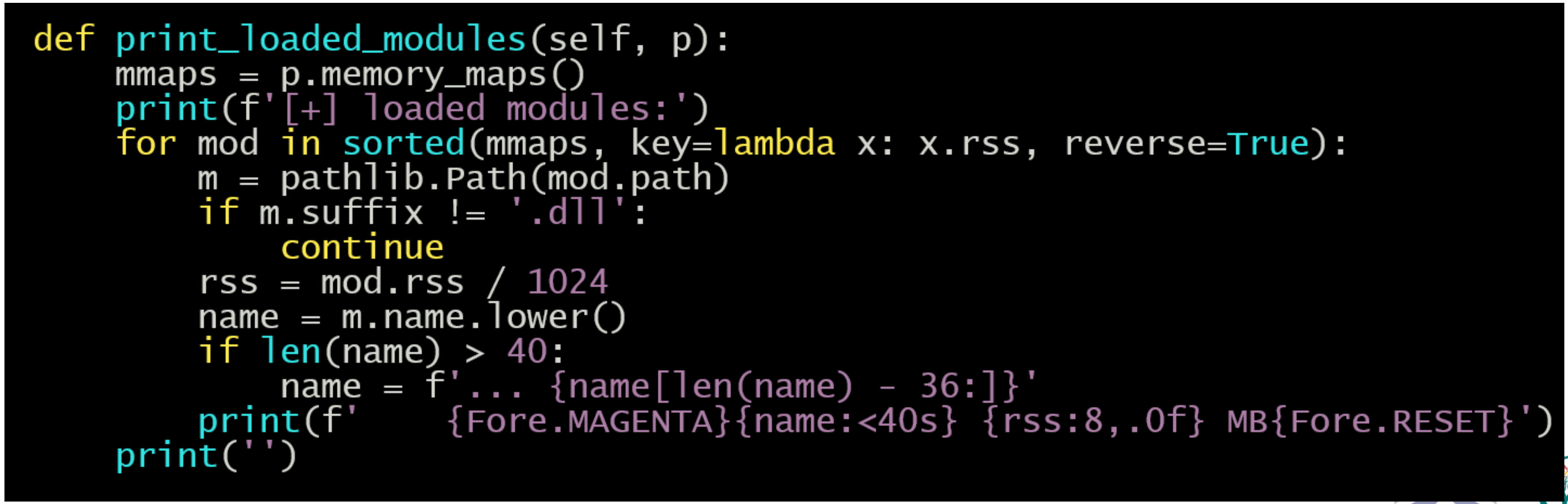
- Within the process listing loop
  - If we want to print command line, then we call a method to do so
  - If we want to print loaded modules, we call a method to do so
  - We also wrap in exception handling code because we might see a process rights failure retrieving more info!



```
try:
    if self.printcmd:
        self.print_cmdline_args(p)
    if self.printmods:
        self.print_loaded_modules(p)
except Exception:
    continue
```

Printing loaded modules

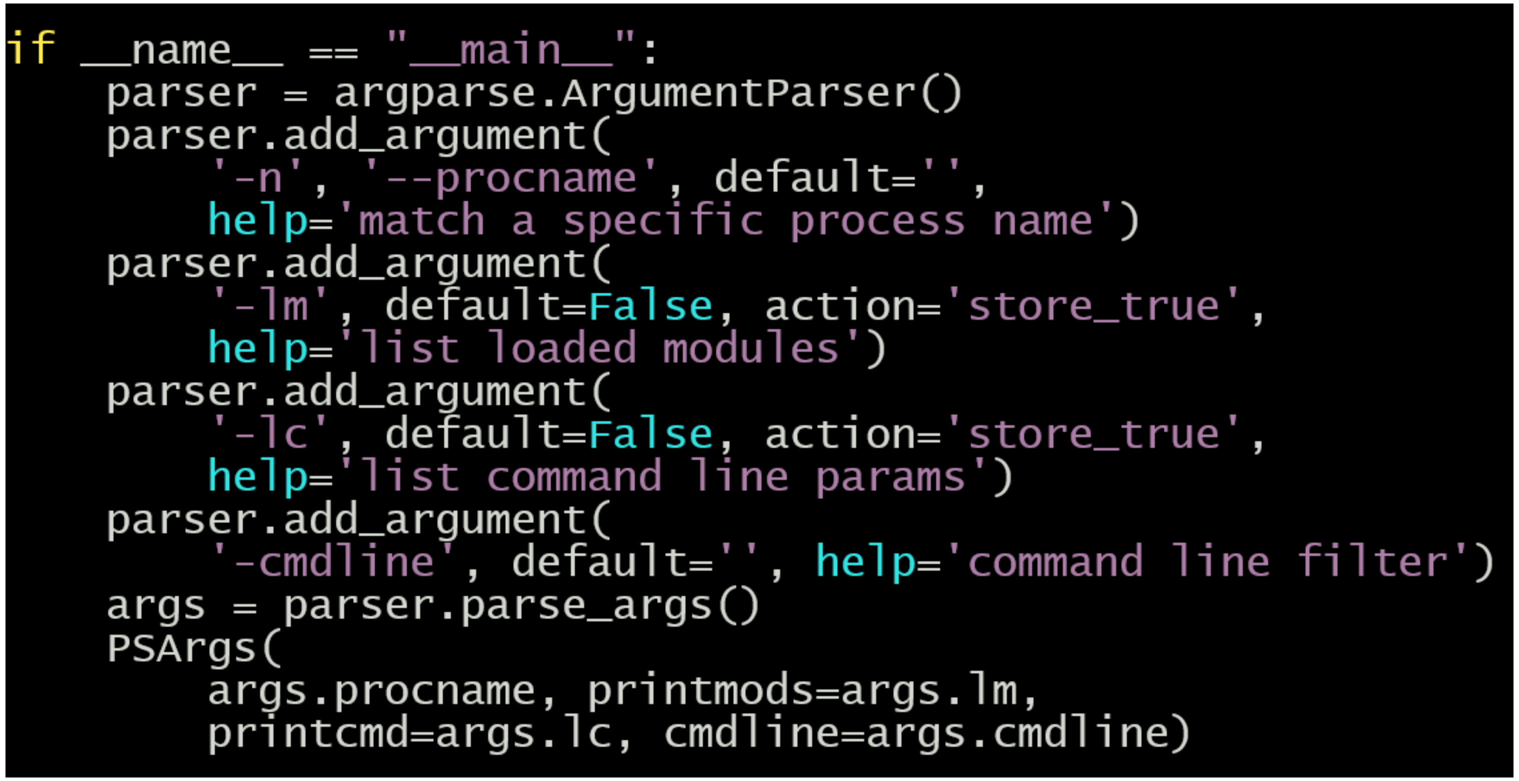
- Here we are using a Python feature called “lambda functions” to impact the sort order
- My goal was to print modules by largest memory footprint first! (rss)



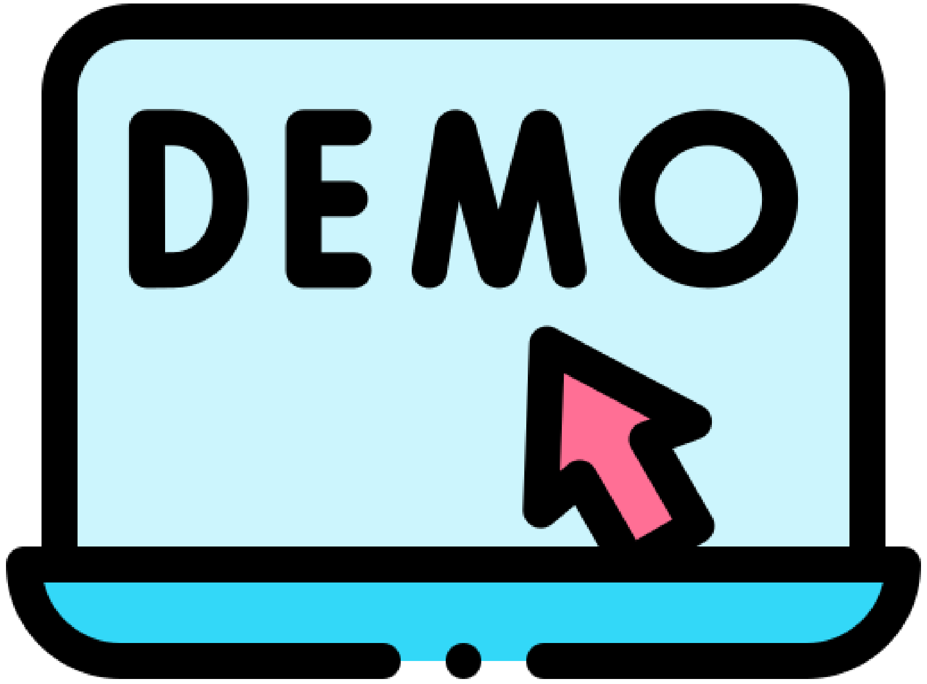
```
def print_loaded_modules(self, p):
    mmaps = p.memory_maps()
    print(f'[+] loaded modules:')
    for mod in sorted(mmaps, key=lambda x: x.rss, reverse=True):
        m = pathlib.Path(mod.path)
        if m.suffix != '.dll':
            continue
        rss = mod.rss / 1024
        name = m.name.lower()
        if len(name) > 40:
            name = f'... {name[len(name) - 36:]}'
        print(f'    {Fore.MAGENTA}{name:<40s} {rss:8,.0f} MB{Fore.RESET}')
    print('')
```

Main part of the script

- Two main sections to this
  - Accept command line arguments
  - Create instance of PSArgs() class with provided parameters



```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '-n', '--procname', default='',
        help='match a specific process name')
    parser.add_argument(
        '-lm', default=False, action='store_true',
        help='list loaded modules')
    parser.add_argument(
        '-lc', default=False, action='store_true',
        help='list command line params')
    parser.add_argument(
        '-cmdline', default='', help='command line filter')
    args = parser.parse_args()
    PSArgs(
        args.procname, printmods=args.lm,
        printcmd=args.lc, cmdline=args.cmdline)
```

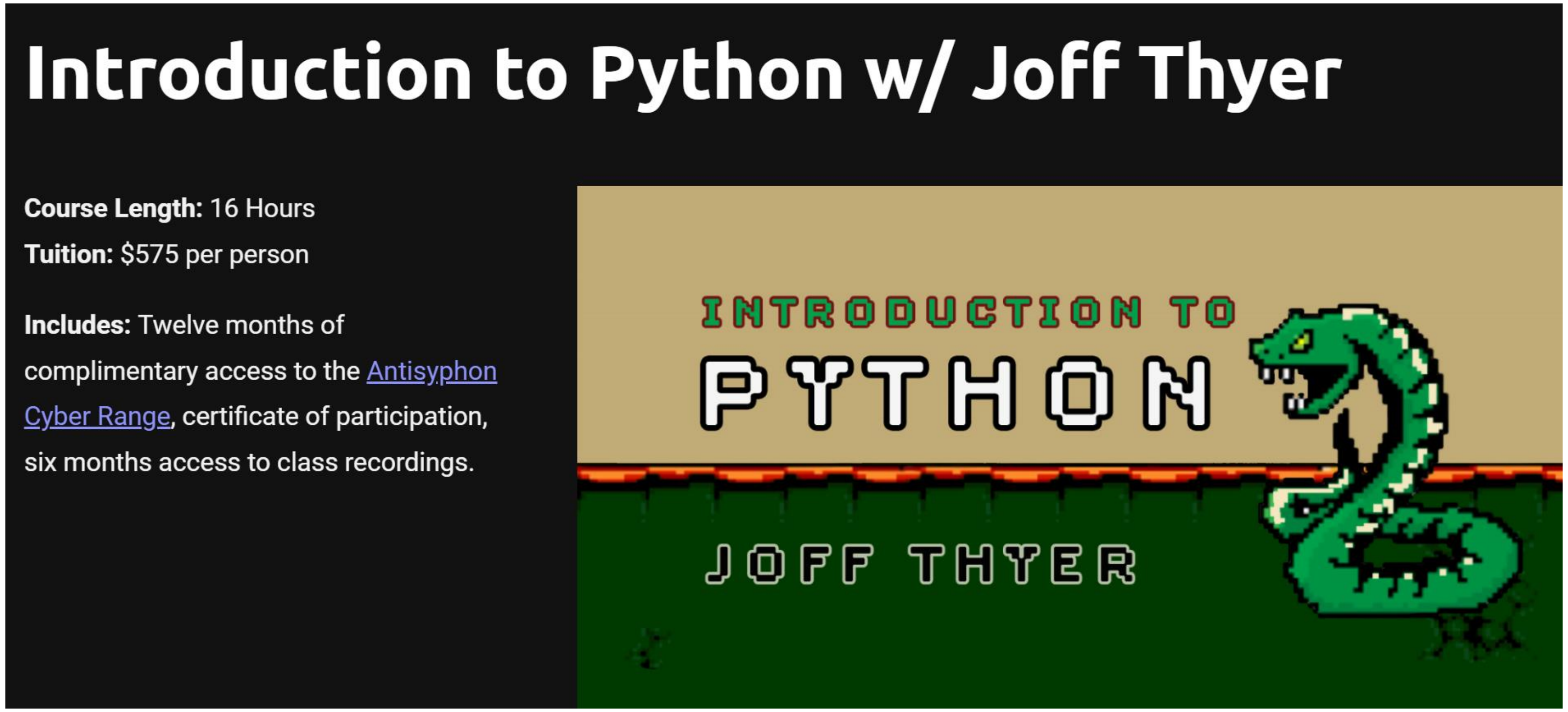


https://github.com/yoda66/ps

Questions?

- https://www.antisyphontraining.com/live-courses-catalog/introduction-to-python-w-joff-thyer/

https://github.com/yoda66/ps


Introduction to Python w/ Joff Thyer

Course Length: 16 Hours

Tuition: $575 per person

Includes: Twelve months of complimentary access to the Antisyphon Cyber Range, certificate of participation, six months access to class recordings.