Python Scripting (MAC CHANGER)

By : Colin Gunsam

LinkedIn: https://www.linkedin.com/in/colingunsam/

Github: https://github.com/infidel101/

Credits : Zaid Sabih and zSecurity (Instructor and his company) where I learnt the following

First we shall be building a simple MAC Address changer, to speed up our process.

Optparse

Usage: Parsing data , for options like x`-v, -i, or even - - help

Example :

- - help

Syntax:

Make an "Object" after importing optparse

Example :

```
import optparse

parser = optparse.OptionParser()
```

The "OptionParser" is the object and "parser" is the variable we have assigned the function of "OptionParser" to.

Setting up Options

Example:

```
import optparse

parser = optparse.OptionParser()
parser.add_option("-i", "--interface", dest="interface",  help="The interface you would like the change." )
```

```
import optparse

parser = optparse.OptionParser()
parser.add_option("-i", "--interface", dest="interface",  help="The interface you would like the change." )
parser.add_option("-m", "--new_mac", dest="new_mac",  help="The new MAC address you would like to change to ." )
```

dest = the destination or variable that will hold the value of anything that comes from the -i or - -interface arguments

help = the help message that will display with the -h or - -help flag.

Parsing the arguments

Example:

```
parser.parse_args()
```

Everything so far:

```
import subprocess #This is for a later part of the script
import optparse

parser = optparse.OptionParser()
parser.add_option("-i", "--interface", dest="interface",  help="The interface you would like the change." )
parser.add_option("-m", "--new_mac", dest="new_mac",  help="The new MAC address you would like to change to ." )
parser.parse_args()
```

parse_args = This is a method that allows the object ("parser") to be able to understand what the user has entered and handle it. **It also returns the arguments and the values** that the user entered to a variable .

Example of running script

```
python3 mac_changer.py --interface eth0 --new_mac 00:11:22:33:44:55
```

The " - - interface " and " - -new_mac" are arguments.

AND

The " eth0 " and " 00:11:22:33:44:55" are values.

In order to capturing these two sets of information (arguments and values) we have to use variables and in order to capture **two** variables on one line we're going to have to use **two** brackets

Example :

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

Everything so far:

```
import optparse
import subprocess


parser = optparse.OptionParser()

parser.add_option("-i", "--interface", dest="interface",  help="The interface you would like the change." )
parser.add_option("-m", "--new_mac", dest="new_mac",  help="The new MAC address you would like to change to ." )

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

Now "options, arguments" are equal to whatever the user input is....

So in other words in order to access the values of "interface" and "new_mac", we must use:

```
 options.interface
```

and

```
options.new_mac
```

as that's where their respective values that we have parsed from the user input are located.

So to set the variables that we can use in our script we would use :

Setting Variables using input from optparse:

```
interface = options.interface
  new_mac = options.new_mac
```

Everything so far:

```
import optparse
import subprocess


parser = optparse.OptionParser()

parser.add_option("-i", "--interface", dest="interface",  help="The interface you would like the change." )
parser.add_option("-m", "--new_mac", dest="new_mac",  help="The new MAC address you would like to change to ." )

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

Now lets introduce a print statement so we can track everything we're doing .

```
print("[+] Changing MAC Address for " + interface + " to " + new_mac ".")
```

Great! Now we can inform the user we are changing the MAC address of <whatever interface they have selected> to <whatever value they want to change the mac address to>

Now lets continue our script and change the MAC Address, now in order to change the MAC Address we have to :

How to change MAC Address Manually :

First we have to disable the interface:

```
ifconfig eth0 down
```

Next we have to change the value of the MAC address :

```
ifconfig eth0 hw ether 00:11:22:33:44:55
```

Finally we have to re-enable the interface:

```
ifconfig eth0 up
```

Now that we understand how to change it manually let's write the script to change it automatically based on the variables the user input.

For this we will have to use the subprocess "call" module, this module allows us to interact with the shell command line.

Example:

```
subprocess.call(["ifconfig", interface, "down"])
subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
subprocess.call(["ifconfig", interface, "up"])
```

Now the reason why everything is separated with commas and the strings are individually quoted is because we are sanitizing our input and prevent users from hijacking our script with malicious code.

Everything so far:

```
#!/usr/bin/env python

import subprocess
import optparse


parser = optparse.OptionParser()

parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")

(options, arguments) = parser.parse_args()

interface = options.interface
new_mac = options.new_mac

print("[+] Changing MAC address for " + interface + " to " + new_mac)

subprocess.call(["ifconfig", interface, "down"])
subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
subprocess.call(["ifconfig", interface, "up"])
```

Cleaning up our code

Lets start by first putting the code into a function, in order to do this we have to define the function name then, specify what input the function takes.

```
def change_mac(interface, new_mac):
  print("[+] Changing MAC address for " + interface + " to " + new_mac)

  subprocess.call(["ifconfig", interface, "down"])
  subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
  subprocess.call(["ifconfig", interface, "up"])
```

change_mac = function name

interface and new_mac = these are the input this functions takes

- Then we hit enter and anything that is indented under that function is part of it

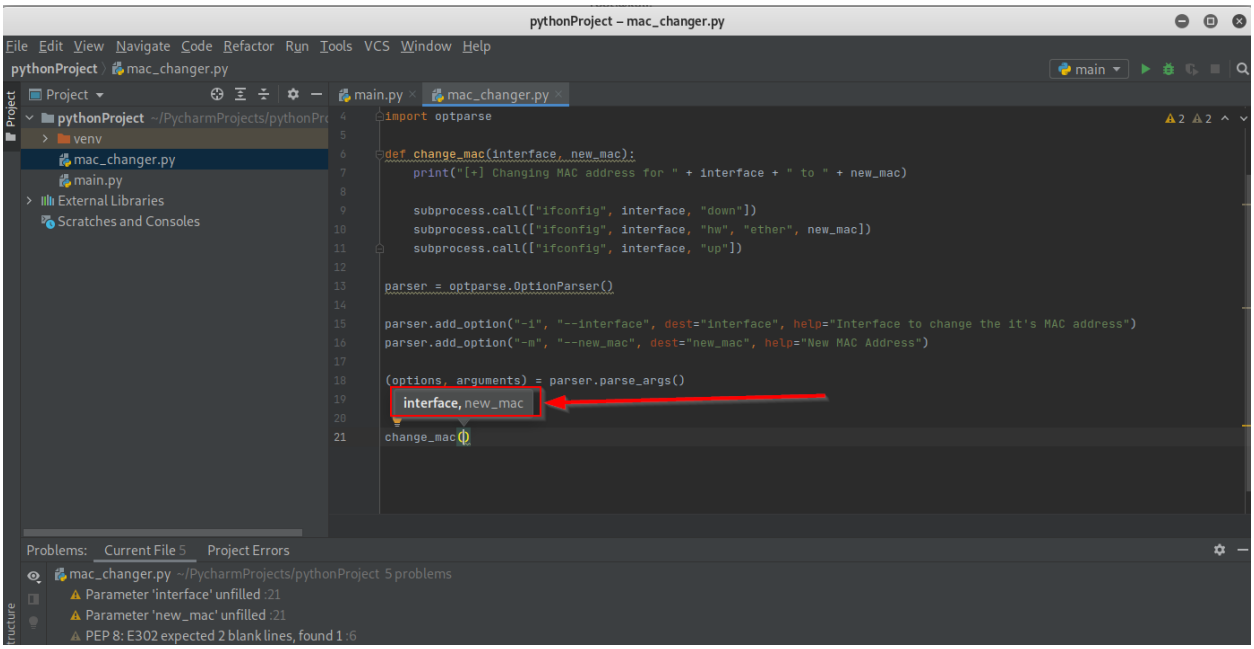
As we can see above we cut the part of the script from the bottom that changes the mac and paste it under the newly created function.

Now make sure to define this function at the beginning of the script. Additionally take note that the function won't work unless we call it and give it the required

```
change_mac(options.interface, options.new_mac)
```

Since we're using the options directly we can remove the variables we set before.

As we can tell it informs us that what we need to input and in what order



So everything so far should look like this:

```
#!/usr/bin/env python

import subprocess
import optparse


def change_mac(interface, new_mac):
    print("[+] Changing MAC address for " + interface + " to " + new_mac)

    subprocess.call(["ifconfig", interface, "down"])
    subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
```

```
    subprocess.call(["ifconfig", interface, "up"])


parser = optparse.OptionParser()

parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")

(options, arguments) = parser.parse_args()

change_mac(options.interface, options.new_mac)
```

Now we want to make te code even cleaner, do the same thing for the rest of it and cut and paste from "parser = " to "options,.." and call it get_argument:

```
def get_arguments():
    parser = optparse.OptionParser()
    parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
    parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")
    (options, arguments) = parser.parse_args()
```

Then what we're going to do since we want to use the value of whatever is inside of get_arguments() is change the part of the function that stores the value of parser.parse_args()

And change that value to "return"

So whenever get_arguments() will return the value of parser.parse_args to the location were it was called from.

So that part of the code should now look like:

```
def get_arguments():
    parser = optparse.OptionParser()
```

```
    parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
    parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")
    return parser.parse_args()
```

And we can capture these results the same way we capture parser_args

by putting

```
(options, arguments) = get_arguments()
```

Now python would know what options is equal too in the change_mac() function.

Now after all is said and done it should look like this:

```
#!/usr/bin/env python

import subprocess
import optparse


def get_arguments():
    parser = optparse.OptionParser()
    parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
    parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")
    return parser.parse_args()


def change_mac(interface, new_mac):
    print("[+] Changing MAC address for " + interface + " to " + new_mac)
    subprocess.call(["ifconfig", interface, "down"])
    subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
    subprocess.call(["ifconfig", interface, "up"])


(options, arguments) = get_arguments()
change_mac(options.interface, options.new_mac)
```

Now that's a lot more tidier!

Now time to add some decision making

Now we want an Error to be given to the user if they don't input an interface or mac address.

So...We have to first move the

```
(options, arguments) =
```

back to the

```
get_arguments()
```

function

So it'll look like

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

Then we're going to introduce and **IF** statement, followed by the conditions

Example:

```
if not options.interface:
  #code to handle this error
elif not options.new_mac
```

This simply means, if options.interface does **NOT** hold a value.

If this is true (options.interface does **NOT** hold a value.) then

it will display the text we put, this can be done with a simple

```
print "Error..."
```

statement

OR

```
parser.error("Error...")
```

After it checks if the options.interface variable does not have a value, in the event it doesn't have a value it will display the error message but if it does contain a value then it will check if the options.new_mac variable does not have a value, in the event that it doesn't have a value it will display the error message and exit the script, but if it does have a value then it will continue to run the rest of the script

Finally return the options inside of the get_arguments() function (OUTSIDE of the if statement)

```
return options
```

Then put capture the options from the get_arguments() function, outside of the function.

```
options = get_arguments()
change_mac(options.interface, options.new_mac)
```

As you can see we stored the values returned by the

```
get_arguments()
```

function inside of the:

```
options
```

variable.

Now the entire script should look like this:

```
#!/usr/bin/env python

import subprocess
import optparse


def get_arguments():
    parser = optparse.OptionParser()
    parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
    parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")
    (options, arguments) = parser.parse_args()
    if not options.interface:
        #Code to handle error
        parser.error("[-]You forgot to include an Interface to change. Use --help for more info")
    elif not options.new_mac:
        #code to handle another error
        parser.error("[-]You forgot to include a new MAC Address. Use --help for more info")
    return options



def change_mac(interface, new_mac):
    print("[+] Changing MAC address for " + interface + " to " + new_mac)
    subprocess.call(["ifconfig", interface, "down"])
    subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
    subprocess.call(["ifconfig", interface, "up"])


options = get_arguments()
change_mac(options.interface, options.new_mac)
```

Now let's check to see if it has been changed:

For this we're going to use the

```
subprocess.check_output()
```

method. Make sure to store the result of the check_output query to a variable :

```
ifconfig_result = subprocess.check_output(["ifconfig", options.interface])
```

Now that we want to sort through the output and search for a particular value, we can use "Regex'.

Navigate to pythex.org . Here we can test the various rules and make sure we are highlighting the correct thing(s).

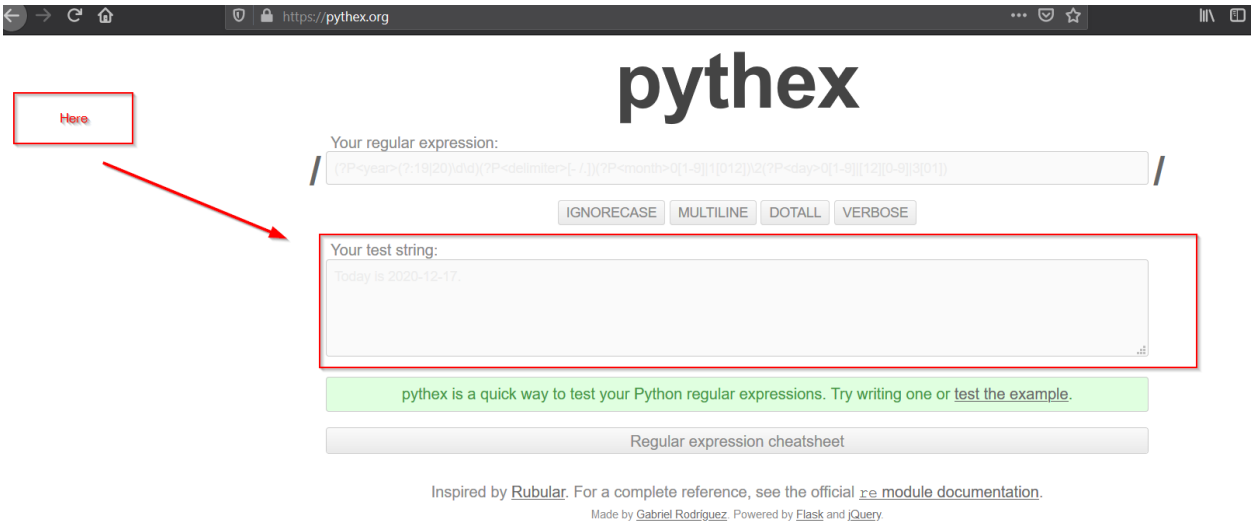
So let's first use the

```
ifconfig eth0
```

command inside of a terminal. Copy it's results

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether 22:33:44:55:66:77  txqueuelen 1000  (Ethernet)
        RX packets 454880  bytes 590149546 (562.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 60428  bytes 7271452 (6.9 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

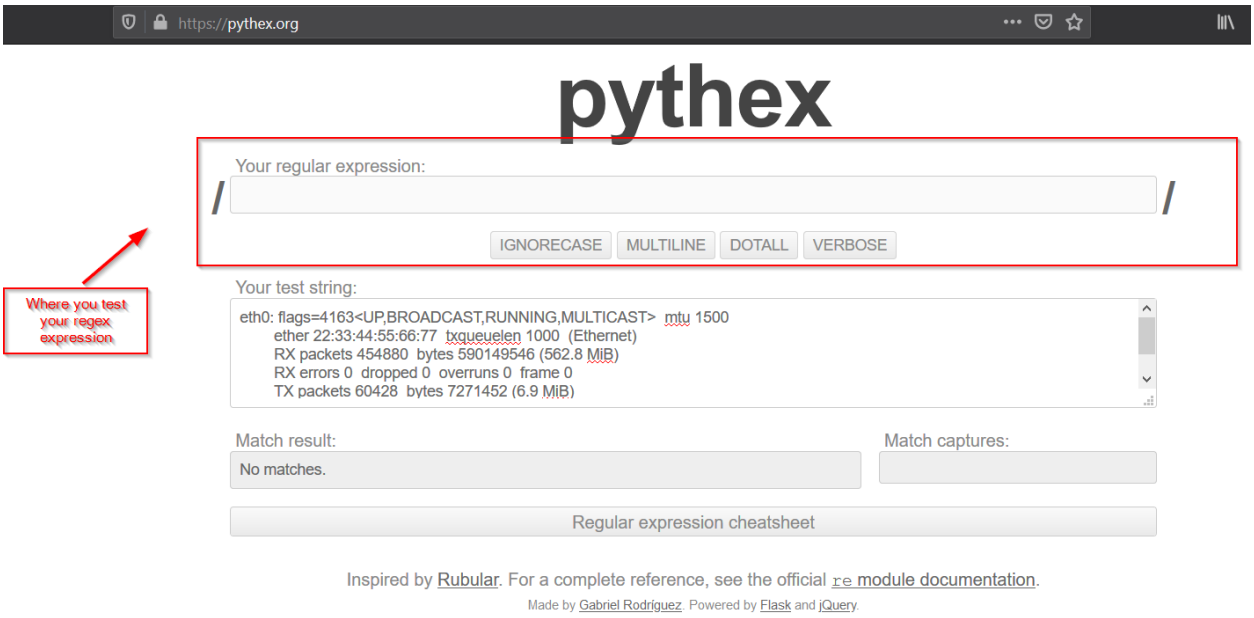
Head over to pythex.org



This is where you paste your results and test them

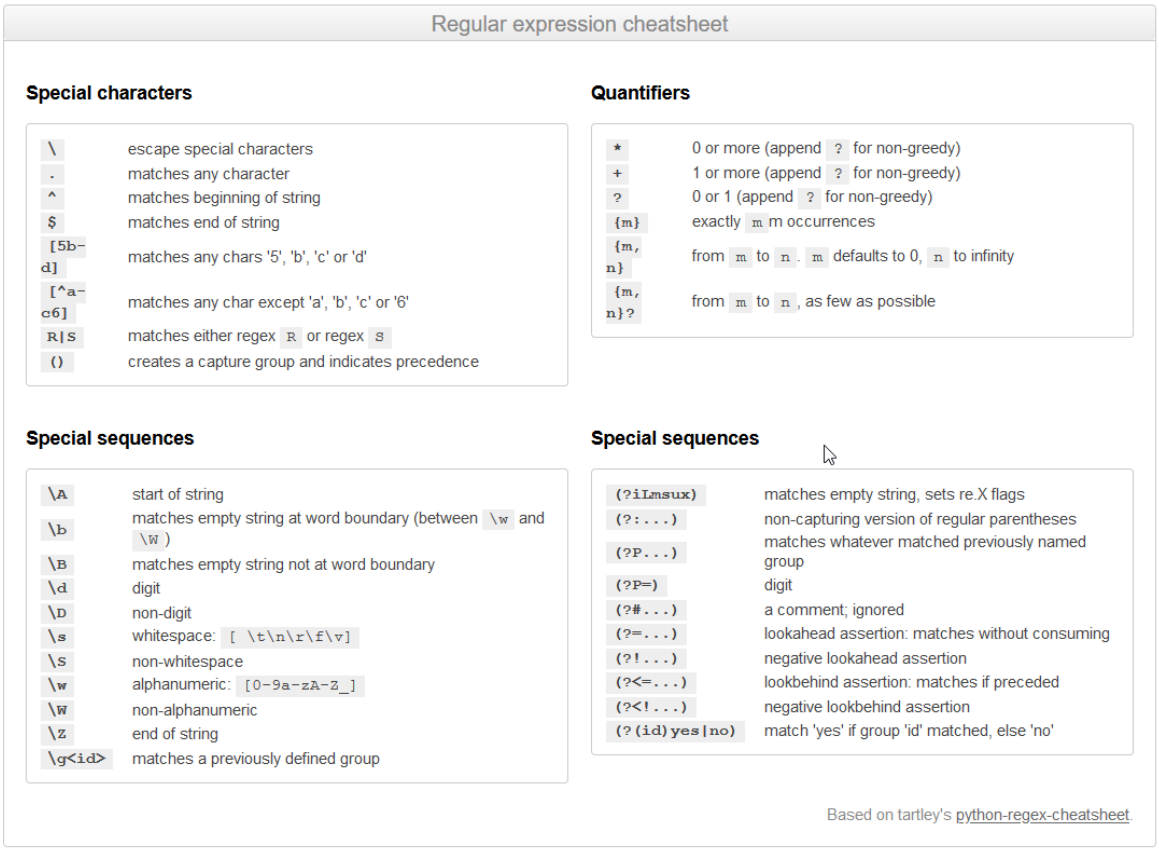
AND

Above where it says "regular expression" is where you can test regex rules (regular expression rules).



So scroll down to the cheat sheet and find out how can we specify the ether parameter in regular expression.

Based on our review of the cheat sheet as seen below:



Regular expression cheatsheet

Special characters

`\`	escape special characters
`.`	matches any character
`^`	matches beginning of string
`$`	matches end of string
`[5b-d]`	matches any chars '5', 'b', 'c' or 'd'
`[^a-c6]`	matches any char except 'a', 'b', 'c' or '6'
`R|S`	matches either regex `R` or regex `S`
`()`	creates a capture group and indicates precedence

Quantifiers

`*`	0 or more (append `?` for non-greedy)
`+`	1 or more (append `?` for non-greedy)
`?`	0 or 1 (append `?` for non-greedy)
`{m}`	exactly `m` m occurrences
`{m, n}`	from `m` to `n`. `m` defaults to 0, `n` to infinity
`{m, n}?`	from `m` to `n`, as few as possible

Special sequences

`\A`	start of string
`\b`	matches empty string at word boundary (between `\w` and `\W`)
`\B`	matches empty string not at word boundary
`\d`	digit
`\D`	non-digit
`\s`	whitespace: `[ \t\n\r\f\v]`
`\S`	non-whitespace
`\w`	alphanumeric: `[0-9a-zA-Z_]`
`\W`	non-alphanumeric
`\Z`	end of string
`\g<id>`	matches a previously defined group

Special sequences

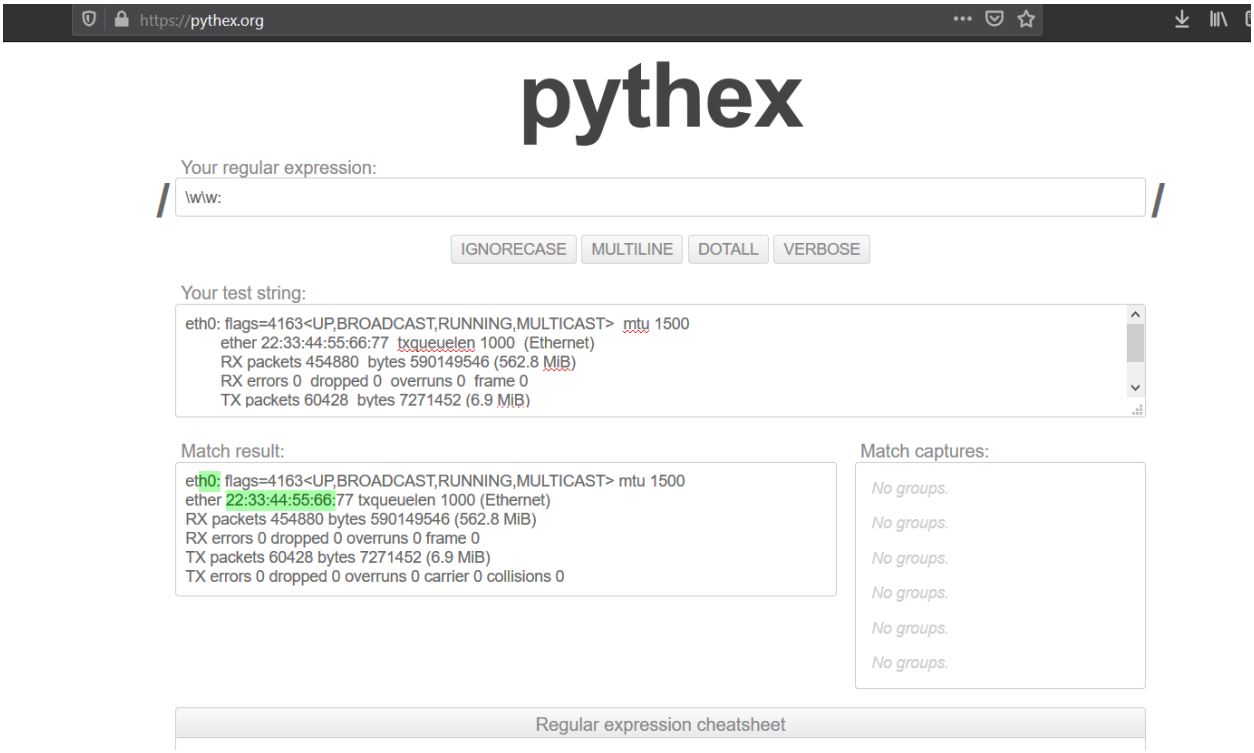
`(?iLmsux)`	matches empty string, sets re.X flags
`(?:...)`	non-capturing version of regular parentheses
`(?P...)`	matches whatever matched previously named group
`(?P=)`	digit
`(?#...)`	a comment; ignored
`(?=...)`	lookahead assertion: matches without consuming
`(?!...)`	negative lookahead assertion
`(?<=...)`	lookbehind assertion: matches if preceded
`(?<!...)`	negative lookbehind assertion
`(?(id)yes|no)`	match 'yes' if group 'id' matched, else 'no'

Based on tartley's python-regex-cheatsheet.

We can see under "Special sequences" that in order to specify alphanumeric characters we have to use

```
/w
```

So lets try it out as we want to specify the MAC address with is made of alphanumeric characters.



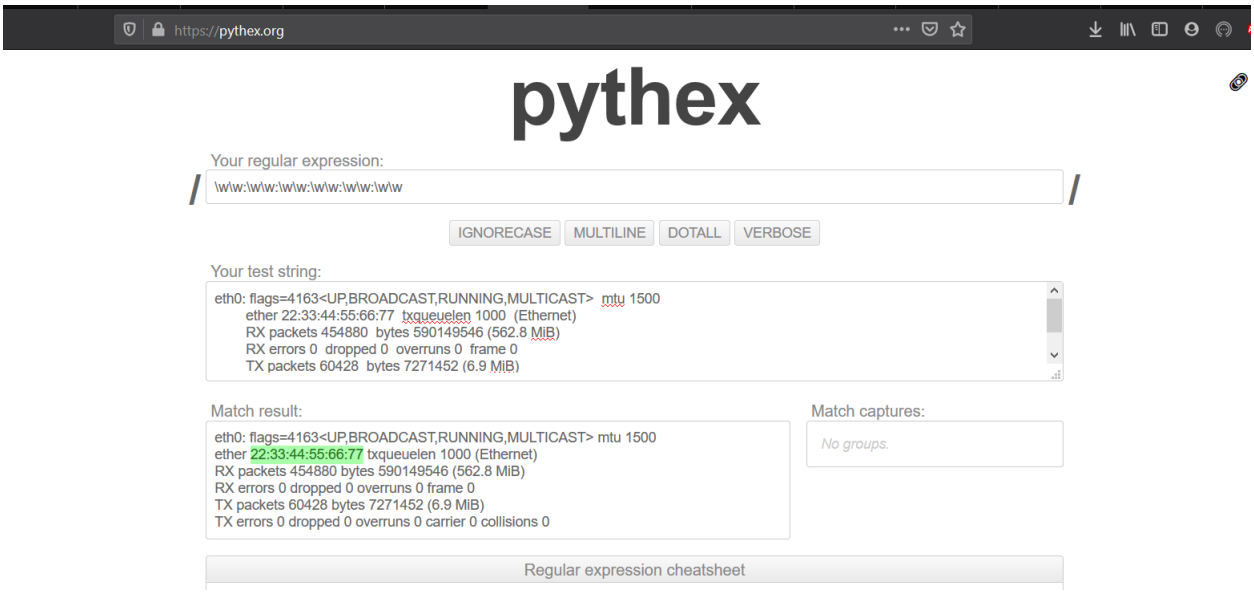
As we can see we put

```
\w\w:
```

It means that we want :

An Alphanumeric character **FOLLOWED** by another Alphanumeric character **FOLLOWED** by a colon

As we can see that specification did narrow our search down to some of our MAC Address and the interface name (we don't want the interface name to be highlighted). So lets narrow it down some more.



Simply by copying and pasting the first set of " \w\w: " until we see our MAC Address is highlighted we discovered that the correct regular expression rules is:

```
\w\w:\w\w:\w\w:\w\w:\w\w:\w\w
```

So continuing ... Lets use the regex rule to extract a substring (the MAC Address)

In order to use regular expressions we need to use the module called "re" so go ahead and import it

```
import re
```

In order to use this you have to use the "search" method

```
re.search()
```

And in order to give a python program a regex rule, you have to first type "r" and use quotation marks and input your rule like a string. Example:

```
re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w")
```

then we have to give it the variable we are searching through to find the pattern, which is the "ifconfig_result" variable.

```
re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w", ifconfig_result)
```

Now please note we are searching for the pattern (our MAC Address) but we're not returning it anyway. So we have to store the results in a variable. In order to capture the result and store it we use the

following syntax:

```
mac_address_search_result =  re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w", ifconfig_result)
```

Then print it

```
print(mac_address_search_result)
```

If when you print it you get multiple queries then, add

```
.group(0)
```

to the query, so it'll look like this:

```
print(mac_address_search_result.group(0))
```

The reason why the number is 0, is because in python when iterating(cycling) through a list (with multiple results) the count begins at 0 rather than 1, so 1 will be the 2nd item in the list.

Next put the above in and IF statement, to provide an error output incase the user types an interface we can't read like "lo" (Virtual network interface...only those of us who are using virtual machines will see this).

So..... The syntax is:

```
if mac_address_search_result:
  print(mac_address_search_result.group(0))
else:
print("[-] Could not read MAC address.")
```

Great our script should look like:

```
#!/usr/bin/env python3.7

import subprocess
import optparse
import re


def get_arguments():
    parser = optparse.OptionParser()
    parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
    parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")
    (options, arguments) = parser.parse_args()
    if not options.interface:
        # Code to handle error
        parser.error("[-]You forgot to include an Interface to change. Use --help for more info")
    elif not options.new_mac:
        # code to handle another error
        parser.error("[-]You forgot to include a new MAC Address. Use --help for more info")
```

```
    return options


def change_mac(interface, new_mac):
    print("[+] Changing MAC address for " + interface + " to " + new_mac)
    subprocess.call(["ifconfig", interface, "down"])
    subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
    subprocess.call(["ifconfig", interface, "up"])


options = get_arguments()
change_mac(options.interface, options.new_mac)

ifconfig_result = subprocess.check_output(["ifconfig", options.interface])
print(ifconfig_result)

mac_address_search_result = re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w:\w\w", ifconfig_result)

if mac_address_search_result:
    print(mac_address_search_result.group(0))
else:
    print("[-] Could not read MAC address.")
```

So far.

Now to tidy it up a bit, First turn the

```
ifconfig_result = subprocess.check_output(["ifconfig", options.interface])
print(ifconfig_result)

mac_address_search_result = re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w:\w\w", ifconfig_result)

if mac_address_search_result:
    print(mac_address_search_result.group(0))
else:
    print("[-] Could not read MAC address.")
```

into function by :

```
def get_current_mac(interface):
    ifconfig_result = subprocess.check_output(["ifconfig", interface])

    mac_address_search_result = re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w:\w\w", ifconfig_result)

    if mac_address_search_result:
        print(mac_address_search_result.group(0))
    else:
        print("[-] Could not read MAC address.")
```

Then call the function so that the code can be executed :

```
get_current_mac(options.interface)
```

The reason why we use "options.interface" when calling the function is because we are storing whatever is the value of options.interface in the interface variable that is in the "get_current_mac()" function. As

such we can now use the code inside of the function get_current_mac() and that code can now access our variable "options.interface" from our "get_arguments()" function.

Now we also have to get the mac address search result from the get_current_mac() function.

So we change the print command to "return" as we're trying to return the output of the query/search if it has a value. If not the error message will print...I'm referring to the code below.

```
if mac_address_search_result:
        print(mac_address_search_result.group(0))
    else:
        print("[-] Could not read MAC address.")
```

Change it to:

```
if mac_address_search_result:
        return(mac_address_search_result.group(0))
    else:
        print("[-] Could not read MAC address.")
```

NOTE : Remove parentheses/brackets "()", if you don't pycharm should tell you "redundant parentheses".

Next make sure to store the value that you are returning, ALWAYS remember if you are return something it must have somewhere to return to.

In order to store this value simply put <variable_that_will_hold_value> = function()

Explanation whenever we use a "return" command we are telling python to return the value (MAC Address in this case) to the line were it was called.

Now finally lets print a statement using this variable to verify whether or not we have successfully changed the MAC address of the intended interface.

So far:

```
#!/usr/bin/env python3.7

import subprocess
import optparse
import re


def get_arguments():
    parser = optparse.OptionParser()
    parser.add_option("-i", "--interface", dest="interface", help="Interface to change the it's MAC address")
    parser.add_option("-m", "--new_mac", dest="new_mac", help="New MAC Address")
    (options, arguments) = parser.parse_args()
    if not options.interface:
        # Code to handle error
        parser.error("[-]You forgot to include an Interface to change. Use --help for more info")
    elif not options.new_mac:
```

```
        # code to handle another error
        parser.error("[-]You forgot to include a new MAC Address. Use --help for more info")
    return options


def change_mac(interface, new_mac):
    print("[+] Changing MAC address for " + interface + " to " + new_mac)
    subprocess.call(["ifconfig", interface, "down"])
    subprocess.call(["ifconfig", interface, "hw", "ether", new_mac])
    subprocess.call(["ifconfig", interface, "up"])


def get_current_mac(interface):
    ifconfig_result = subprocess.check_output(["ifconfig", interface])

    mac_address_search_result = re.search(r"\w\w:\w\w:\w\w:\w\w:\w\w:\w\w:\w\w", ifconfig_result)

    if mac_address_search_result:
        return mac_address_search_result.group(0)
    else:
        print("[-] Could not read MAC address.")


options = get_arguments()
change_mac(options.interface, options.new_mac)
current_mac = get_current_mac(options.interface)
print("The " + options.interface + " new MAC Address is " + current_mac)
```

Now we have to cast (Treating a different data type as a different data type)

example : treating a none-object as a string

```
str(100)
```

This basic example will use 100 as a string.

You only have to cast if you are not getting proper results during testing, example of how to cast:

```
print("The " + options.interface + " new MAC Address is " + str(current_mac))
```

Now finally to check to see if the MAC Address was successfully changed, we shall use a simple IF statement

```
if current_mac == options.new_mac:
  print("[+] MAC Address was successfully changed to " + options.new_mac)
else:
  print("[-] MAC Address was NOT successfully changed.")
```

Now what this does is checks to see if the current MAC Address is equal to the one the user requested, if it is it will inform the user it has be successfully changed, however if it isn't changed, it will all so inform the user that it was unsuccessful.