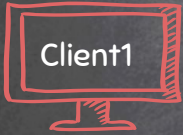


MAC ADDRESS

- Media Access Control
 - Permanent.
 - Physical.
 - Unique.
- Assigned by manufacturer.

WHY CHANGE THE MAC ADDRESS?

1. Increase **anonymity**.
2. **Impersonate** other devices.
3. **Bypass** filters.



Mac = 00:11:11:11:11:11

Access Point



Mac = 00:22:22:22:22:22

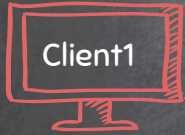


Resources
eg:internet



Mac = 00:11:22:33:44:55

Source Mac: 00:11:11:11:11:11
Destination Mac: 00:22:22:22:22:22



Client1



Mac = 00:11:11:11:11:11

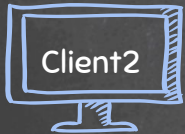
Access Point



Mac = 00:22:22:22:22:22



Resources
eg:internet



Client2

Mac = 00:11:22:33:44:55

MAC_CHANGER

USING A MODULE TO EXECUTE SYSTEM COMMANDS

- The **subprocess** module contains a number of functions.
- These functions allow us to **execute system commands**.
- Commands depend on the **OS** which executes the script.

Syntax:

```
import subprocess  
subprocess.call("COMMAND", Shell=True)
```



MAC_CHANGER

VARIABLES

- A variable is a location in memory that contains a certain value.
- Similar to maths, its a name that is used to store information.

Ex:

`X = 1`

Now x has a value of one, so we can do

`Y = x + x`

And y has a value of 2 now.

`print(y)`

Will print the value of y on screen which is 2.



MAC_CHANGER

HANDLING USER INPUT

- Easiest way of getting user input is through keyboard.
- There are a number of ways to achieve that.
- `input()` function prompts the user to enter a value.

Ex:

```
age = input("What is your age?")
```

Result

What is your age ?

The variable `age` will hold the value of the user input



MAC_CHANGER

FUNCTIONS

- Set of instructions to carry out a task.
- Can take input, and return result.
- Make the code clearer, reusable, and more abstract.
- `input()` function prompts the user to enter a value.

Ex, we can define a function like so:

```
def function_name(variable1, variable2 ...etc)
```

And call it in code like so:

```
function_name(value1, value2)
```



MAC_CHANGER

DECISION MAKING

- Execute code ONLY if a condition is true.

```
if condition:
    #Code to execute when
    #condition is true
else:
    #Code to execute when
    #condition is false

#Rest of code
```

```
if condition1:
    #Code to execute when
    #condition1 is true
elif condition2:
    #Code to execute when
    #condition2 is true AND
    #condition 1 is false
else:
    #Code to execute when
    #ALL conditions are FALSE

#Rest of code
```

<https://t.me/learningnets>

```
if condition1:
    #Code to execute when
    #condition1 is true
if condition2:
    #Code to execute when
    #condition2 is true

#Rest of code
```

MAC_CHANGER

SIMPLE ALGORITHM

Goal → Check if MAC address was changed.

Setps:

1. Execute and read ifconfig.
2. Read the mac address from output.
3. Check if MAC in ifconfig is what the user requested.
4. Print appropriate message.



MAC_CHANGER

SIMPLE ALGORITHM

Goal → Check if MAC address was changed.

Setps:

1. Execute and read ifconfig.
2. Read the mac address from output.
3. Check if MAC in ifconfig is what the user requested.
4. Print appropriate message.



MAC_CHANGER

SIMPLE ALGORITHM

Goal → Check if MAC address was changed.

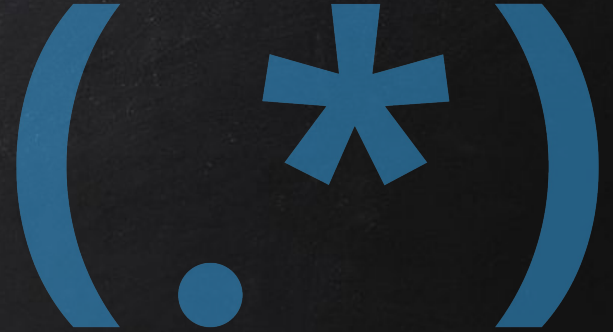
Setps:

1. Execute and read ifconfig.
2. Read the mac address from output.
3. Check if MAC in ifconfig is what the user requested.
4. Print appropriate message.



MAC_CHANGER

REGULAR EXPRESSIONS



- Search for specific **patterns** within a string.
- Uses **rules** to match pattern.

→ Great to tell a program what to look for in a large text.

MAC_CHANGER

SIMPLE ALGORITHM

Goal → Check if MAC address was changed.

Setps:

1. Execute and read ifconfig.
2. Read the mac address from output.
3. Check if MAC in ifconfig is what the user requested.
4. Print appropriate message.



MAC_CHANGER

SIMPLE ALGORITHM

Goal → Check if MAC address was changed.

Setps:

1. Execute and read ifconfig.
2. Read the mac address from output.
3. Check if MAC in ifconfig is what the user requested.
4. **Print appropriate message.**

