

Projet UE 312 Technologies réseaux

**Conception d’une application python capable d’effectuer diverses opérations sur des adresses IPV4**

Développé par

**Popa Stefan**

**Fabio Di Vito**

**Année académique 2022-2023**

Table des matières

[Mode d’emploi 4](#_Toc116586009)

[Lancer le programme 4](#_Toc116586010)

[Tests effectués 7](#_Toc116586011)

[« Trouver un masque via une IP » 7](#_Toc116586012)

[« Trouver le réseau d’une IP » 8](#_Toc116586013)

[« Trouver l’appartenance d’une IP » 9](#_Toc116586014)

[« Appartenance réseau de 2 machines » 10](#_Toc116586015)

[Répartition du travail 11](#_Toc116586016)

[Trello 11](#_Toc116586017)

[Code source 12](#_Toc116586018)

[« Trouver un masque via une IP » 12](#_Toc116586019)

[« Trouver le réseau d’une IP » 13](#_Toc116586020)

[« Trouver l’appartenance d’une IP » 14](#_Toc116586021)

[« Appartenance réseau de 2 machines » 16](#_Toc116586022)

[« Login » 18](#_Toc116586023)

## Mode d’emploi

### Lancer le programme

Pour lancer le programme, il faut lancer le fichier **« Login.py ».**

Une fois le fichier lancé, des instructions apparaîtront dans la console de commande.

Il faudra se connecter à une base de données locale pour accéder aux fonctionnalités du programme.  
Un compte est déjà présent (Nom = « root » Mdp = « ilovepython »)

Une image contenant table

Description générée automatiquementUne fois la connexion effectuée, vous serez renvoyé vers une page web.

Un menu se présentera alors, et une liste de boutons pourra être cliquée pour accéder aux différentes fonctionnalités.

Voici un bref résumé des fonctionnalités qui seront détaillées dans la suite du document :

Trouver un masque via une IP 🡺 En classfull, trouve la classe d’une IP et ses caractéristiques.

Trouver le réseau d’une IP 🡺 En classfull, trouve les informations du réseau d’une IP/masque.

Trouver l’appartenance d’une IP 🡺 En classless, détermine si une IP appartient à un réseau.

Appartenance réseau de 2 machines 🡺 En classless, détermine si 2 machines se considèrent dans le même réseau ou non.

Calculer un nombre d’hôtes 🡺 En classless, détermine si une adresse peut accueillir un certain nombre de sous réseaux et d’hôtes.

Documentation 🡺 Lance la lecture d’un fichier PDF contenant des informations relatives au programme.

* **« Trouver un masque via une IP »** est une fonctionnalité classfull qui permets de déterminer la classe d’une adresse IP entrée et ses caractéristiques (masque, hôtes, …).

Il suffit de rentrer les 4 octets de l’adresse IP et appuyer sur confirmer pour recevoir un résultat.

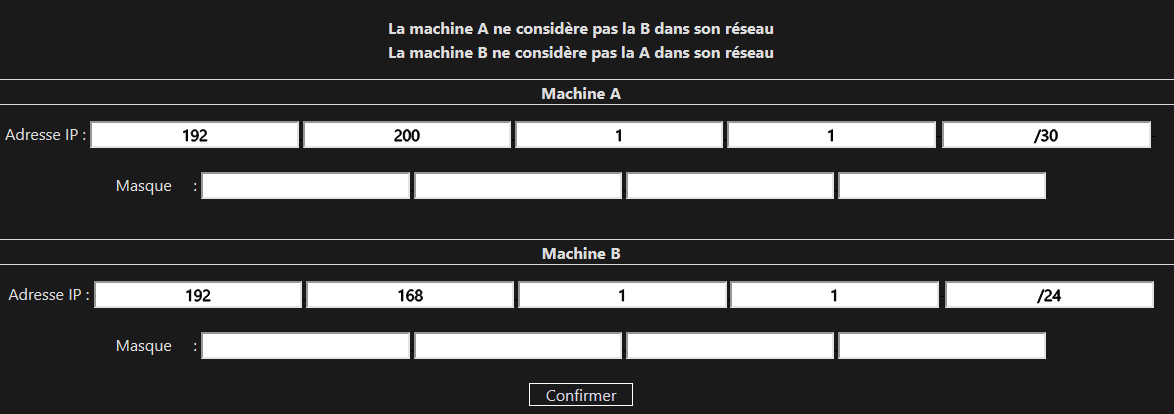
* **Une image contenant texte, capture d’écran, noir, écran

  Description générée automatiquement« Trouver le réseau d’une IP »** est une fonctionnalité classfull qui permets de trouver l’adresse du réseau, l’adresse de broadcast du réseau et l’adresse du sous réseau de l’IP/masque machine entré.

Il suffit d’entrer les 4 octets du masque et les 4 octets de l’IP machine et le résultat sera affiché.

* **« Trouver l’appartenance d’une IP »** est une fonctionnalité classless qui permets de déterminer si une IP/masque machine entré appartiens au réseau IP entré.

Il faut entrer les 4 octets de l’adresse IP machine, ensuite encoder un masque CIDR ou un masque en format 4 octets (si les 2 sont encodés, le masque CIDR sera choisi en priorité), encoder une IP réseau et appuyer sur « confirmer » pour recevoir le résultat.

* **« Appartenance réseau de 2 machines »** est une fonctionnalité classless qui permets de déterminer si une IP/masque machine entré considère une autre IP/masque machine entré dans son réseau ou non, et inversement.

Il faut encoder une adresse IP au format 4 octets et un masque machine format CIDR ou format 4 octets pour chaque machine, et ensuite appuyer sur le bouton confirmer pour voir le résultat.

* **« Calculer un nombre d’hôtes »** est une fonctionnalité classless qui permets de déterminer si une adresse réseau peut accueillir un nombre de sous réseaux / hôtes entrés.



Il faut encoder l’adresse réseau en format 4 octets, et ensuite entrer le nombre de sous réseaux souhaités et le nombre d’hôtes par sous réseau souhaités. Il faut ensuite appuyer sur « confirmer » et le résultat apparaîtra.

## Tests effectués

### « Trouver un masque via une IP »

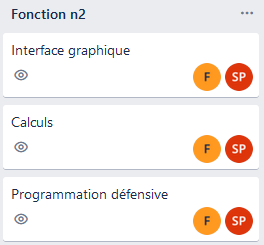
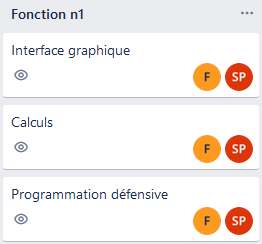
### Une image contenant texte, capture d’écran, moniteur, écran Description générée automatiquement« Trouver le réseau d’une IP »Une image contenant texte Description générée automatiquementUne image contenant texte, capture d’écran, moniteur, noir Description générée automatiquement

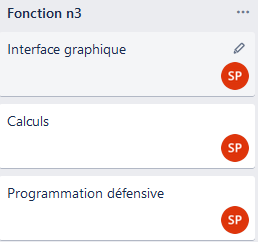
### Une image contenant texte, capture d’écran, moniteur, écran Description générée automatiquementUne image contenant texte, capture d’écran, moniteur, écran Description générée automatiquement« Trouver l’appartenance d’une IP »

### « Appartenance réseau de 2 machines »

## Répartition du travail

### Trello





Une image contenant texte

Description générée automatiquement

## Code source

Le programme a été réalise en Pyscript (Python dans le navigateur) et l’interface graphique et html/CSS.

### « Trouver un masque via une IP »

<py-script>  
 import js  
 def ClassfullClassFinder():  
 try:  
 adresseIP = Element('o1').element.value + " " + Element('o2').element.value +" "+ Element('o3').element.value +" "+ Element('o4').element.value  
 isValid = True  
 octets = adresseIP.split()  
 if(len(octets) != 4):  
 isValid = False  
 for octet in octets:  
 if(int(octet)>255 or int(octet)<0):  
 isValid = False  
 o = octets[0]  
 octets[0] = int(o)  
 texte = ""  
 octets[0] = int(octets[0])  
 if(octets[0]==127 or octets[0]==0):  
 texte="Classe reservee"  
 elif(octets[0]>=0 and octets[0]<=127):  
 texte = "Classe A - Masque 255.0.0.0 - Avec une classe d'adresse A en classfull, on peut concevoir 128 réseaux de 16 777 214 machines"  
 elif(octets[0]>=128 and octets[0]<=191):  
 texte = "Classe B - Masque 255.255.0.0 - Avec une classe d'adresse B en classfull, on peut concevoir 16384 réseaux de 65 534 machines"  
 elif(octets[0]>=192 and octets[0]<=223):  
 texte = "Classe C - Masque 255.255.255.0 - Avec une classe d'adresse C en classfull, on peut concevoir 2 097 152 réseaux de 254 machines"  
 elif(octets[0]>=224 and octets[0]<=239):  
 texte = "Classe D - Pas de masque - Plage d'adresses reservée (multicast)"  
 elif(octets[0]>=240 and octets[0]<=255):  
 texte = "Classe E - Pas de masque - Plage d'adresses reservée (expériences protocoles)"  
 if(isValid):  
 Element("result").write(texte)  
 else:  
 Element("result").write("Adresse IP invalide")  
 except:  
 Element("result").write("Adresse IP invalide")  
</py-script>

### « Trouver le réseau d’une IP »

<py-script>  
 import js  
 import ipaddress  
 import re  
 from ipaddress import IPv4Interface  
 #Vérifier l'intégrité d'un masque classique  
 def verifier\_intergrite\_masque\_classique(masque):  
 masqueValide = True  
 maskInString = str(masque)  
 if (re.search('[a-zA-Z\_]',maskInString)):  
 masqueValide = False  
 return masqueValide  
 masqueV = masque.split()  
 print(masqueV)  
 if(len(masqueV)!=4 or maskInString.replace(" ","")==""):  
 masqueValide = False  
 return masqueValide  
 for octet in masqueV:  
 if (int(octet)>255 or int(octet) <0):  
 masqueValide = False  
 return masqueValide  
 x = range(1,4)  
 for n in x:  
 if(int(masqueV[n])>int(masqueV[n-1])):  
 masqueValide = False  
 return masqueValide  
 return masqueValide  
 def trouver\_Broadcast(IP,Masque):  
 IP = IP.replace(" ",".")  
 ipn= ipaddress.ip\_network(str(IP)+"/"+str(Masque),strict=False)  
 return ipn.broadcast\_address  
 def getNotationCIDR(masque):  
 compteur = 0  
 masqueDivise = masque.split()  
 for octet in masqueDivise:  
 octetBinare = bin(int(octet))[2:]  
 for bit in octetBinare:  
 if(bit == '1'):  
 compteur = compteur +1  
 return compteur  
 def trouver\_Adresse\_Reseau(IP,Masque):  
 IP = IP.replace(" ",".")  
 ifc = IPv4Interface(str(IP)+"/"+str(Masque))  
 return(ifc.network)  
 def trouver\_SR(IP,Masque):  
 IP = IP.replace(" ",".")  
 if(Masque==0 or Masque==8 or Masque==16 or Masque==24 or Masque==32):  
 return("pas de sous réseau")  
 ipn = ipaddress.ip\_network(str(IP)+"/"+str(Masque),strict=False)  
 return (ipn)  
 def trouver\_appartenance():  
 IP\_Machine\_A = Element('o1').element.value + " " + Element('o2').element.value +" "+ Element('o3').element.value +" "+ Element('o4').element.value  
 Masque\_Machine\_A = Element('m1').element.value + " " + Element('m2').element.value +" "+ Element('m3').element.value +" "+ Element('m4').element.value  
 try:  
 if(verifier\_intergrite\_masque\_classique(Masque\_Machine\_A)==True):  
 Masque\_Machine\_A = getNotationCIDR(Masque\_Machine\_A)  
 res1 = trouver\_Broadcast(IP\_Machine\_A,Masque\_Machine\_A)  
 res2 = trouver\_Adresse\_Reseau(IP\_Machine\_A,Masque\_Machine\_A)  
 res3 = trouver\_SR(IP\_Machine\_A,Masque\_Machine\_A)  
 else:  
 Element("result").write("Masque entré invalide")  
 Element("result2").write("")  
 Element("result3").write("")  
 Element("result2").write("Adresse du réseau : "+str(res2))  
 Element("result").write("Adresse de broadcast : "+str(res1))  
 Element("result3").write("Adresse du SR : "+str(res3))  
 except:  
 Element("result").write("Données invalides")  
 Element("result2").write("")  
 Element("result3").write("")  
</py-script>

### « Trouver l’appartenance d’une IP »

<py-script>  
 import js  
 import ipaddress  
 def trouver\_appartenance():  
 IPMachine = Element('o1').element.value + " " + Element('o2').element.value +" "+ Element('o3').element.value +" "+ Element('o4').element.value  
 MasqueMachine = Element('m1').element.value + " " + Element('m2').element.value +" "+ Element('m3').element.value +" "+ Element('m4').element.value  
 MasqueMachineCIDR = Element('CIDR').element.value  
 IPReseau = Element('n1').element.value + " " + Element('n2').element.value +" "+ Element('n3').element.value +" "+ Element('n4').element.value  
 valid = True  
 valid = check\_validite\_IP(IPMachine,MasqueMachine,MasqueMachineCIDR)  
 if(valid==False):  
 print("IP entrée / masque CIDR entré erronés")  
 if(len(MasqueMachine)>3):  
 print("Prise en charge du masque classique")  
 valid = verifierIntegriteMasque(MasqueMachine)  
 MasqueMachine=getNotationCIDR(MasqueMachine)  
 MasqueMachine = "/",str(MasqueMachine)  
 elif(len(MasqueMachineCIDR)==3):  
 print("Prise en charge du masque CIDR")  
 MasqueMachine=MasqueMachineCIDR  
 IPMachine = IPMachine.replace(" ",".")  
 IPReseau = IPReseau.replace(" ",".")  
 try:  
 IPMachine = IPMachine+MasqueMachine[0]+MasqueMachine[1]+MasqueMachine[2]  
 except:  
 IPMachine = IPMachine+MasqueMachine[0]+MasqueMachine[1]  
 try:  
 ipn1 = ipaddress.ip\_network(IPMachine,strict=False)  
 ipn2 = ipaddress.ip\_network(IPReseau)  
 print(ipn1)  
 print(ipn2)  
 if(ipn1.overlaps(ipn2) == True and valid == True):  
 Element("result").write("L'IP appartiens au réseau")  
 elif(valid==True):  
 Element("result").write("L'IP n'appartiens pas au réseau")  
  
 if(valid == False):  
 Element("result").write("Données erronées")  
 except:  
 Element("result").write("Données erronées - Erreur dans les IP/masque")  
 def getNotationCIDR(masque):  
 compteur = 0  
 try:  
 masqueDivise = masque.split()  
 for octet in masqueDivise:  
 octetBinare = bin(int(octet))[2:]  
 for bit in octetBinare:  
 if(bit == '1'):  
 compteur = compteur +1  
 except:  
 Element("result").write("Données erronées")  
 return compteur  
 def check\_validite\_IP(IPMachine, MasqueMachine, MasqueMachineCIDR):  
 valid = True  
 if(MasqueMachine.replace(" ","")=="" and MasqueMachineCIDR.replace(" ","")==""):  
 valid = False  
 MasqueMachineDecoupe = MasqueMachine.split()  
 if(MasqueMachine != "" and MasqueMachineCIDR ==""):  
 for o in MasqueMachineDecoupe:  
 if(int(o) > 255 or int(o)<0):  
 valid = False  
 else:  
 MasqueMachineCIDR = MasqueMachineCIDR.replace("/","")  
 if(int(MasqueMachineCIDR) > 32 or int(MasqueMachineCIDR)<0):  
 valid = False  
 return valid  
 def verifierIntegriteMasque(masque):  
 masqueValide = True  
 masque = masque.split()  
 if(len(masque)!=4):  
 masqueValide = False  
 return masqueValide  
 if(int(masque[0])==0):  
 masqueValide = False  
 return masqueValide  
 for octet in masque:  
 if (int(octet)>255 or int(octet) <0):  
 masqueValide = False  
 return masqueValide  
 x = range(1,4)  
 for n in x:  
 if(int(masque[n])>int(masque[n-1])):  
 masqueValide = False  
 return masqueValide  
</py-script>

### « Appartenance réseau de 2 machines »

<py-script>  
 import ipaddress  
 import re  
 #Vérifier l'intégrité d'un masque classique  
 def verifier\_intergrite\_masque\_classique(masque):  
 masqueValide = True  
 maskInString = str(masque)  
 if (re.search('[a-zA-Z\_]',maskInString)):  
 masqueValide = False  
 return masqueValide  
 masqueV = masque.split()  
 print(masqueV)  
 if(len(masqueV)!=4 or maskInString.replace(" ","")==""):  
 masqueValide = False  
 return masqueValide  
 for octet in masqueV:  
 if (int(octet)>255 or int(octet) <0):  
 masqueValide = False  
 return masqueValide  
 x = range(1,4)  
 for n in x:  
 if(int(masqueV[n])>int(masqueV[n-1])):  
 masqueValide = False  
 return masqueValide  
 #Vérifier l'intégrité d'un masque CIDR  
 def verifier\_integrite\_masque\_CIDR(masque):  
 valid = True  
 masque\_decoupe = str(masque).replace("/","")  
 masque\_decoupe = int(masque\_decoupe)  
 if(masque\_decoupe<0 or masque\_decoupe >32):  
 return False  
 return masqueValide  
 #Transforme un masque classique en masque CIDR  
 def getNotationCIDR(masque):  
 compteur = 0  
 masqueDivise = masque.split()  
 for octet in masqueDivise:  
 octetBinare = bin(int(octet))[2:]  
 for bit in octetBinare:  
 if(bit == '1'):  
 compteur = compteur +1  
 return "/"+str(compteur)  
 def trouver\_masque(masque\_classique,masque\_CIDR):  
 if(masque\_classique=="" and masque\_CIDR!=""):  
 print("Prise en charge du masque CIDR")  
 return masque\_CIDR  
 elif(masque\_CIDR=="" and masque\_classique!=""):  
 print("Prise en charge du masque classique")  
 print(masque\_classique)  
 if(verifier\_intergrite\_masque\_classique(masque\_classique)==True):  
 print("masque valide")  
 return getNotationCIDR(masque\_classique)  
 else:  
 print("masque non valide")  
 return False  
 elif(masque\_classique!="" and masque\_CIDR!=""):  
 print("Double masque - prise du masque CIDR")  
 return masque\_CIDR  
 def getConsidereDansLeReseau(IP1,masque1,IP2,masque2):  
 IP1 = IP1.replace(" ",".")  
 IP2 = IP2.replace(" ",".")  
 ipn1 = ipaddress.ip\_network(str(IP1)+""+str(masque1),strict=False)  
 ipn2 = ipaddress.ip\_network(str(IP2)+""+str(masque2),strict=False)  
 return(ipn1.subnet\_of(ipn2))  
 def TrouverReseauMachines():  
 try:  
 IP\_Machine\_A = Element('o1a').element.value + " " + Element('o2a').element.value +" "+ Element('o3a').element.value +" "+ Element('o4a').element.value  
 Masque\_Machine\_A = Element('m1a').element.value + " " + Element('m2a').element.value +" "+ Element('m3a').element.value +" "+ Element('m4a').element.value  
 Masque\_CIDR\_Machine\_A = Element('CIDRa').element.value  
 IP\_Machine\_B = Element('o1b').element.value + " " + Element('o2b').element.value +" "+ Element('o3b').element.value +" "+ Element('o4b').element.value  
 Masque\_Machine\_B = Element('m1b').element.value + " " + Element('m2b').element.value +" "+ Element('m3b').element.value +" "+ Element('m4b').element.value  
 Masque\_CIDR\_Machine\_B = Element('CIDRb').element.value  
 if(getConsidereDansLeReseau(IP\_Machine\_A,trouver\_masque(Masque\_Machine\_A,Masque\_CIDR\_Machine\_A),IP\_Machine\_B,trouver\_masque(Masque\_Machine\_B,Masque\_CIDR\_Machine\_B))):  
 Element("result").write("La machine A considère la B dans son réseau")  
 else:  
 Element("result").write("La machine A ne considère pas la B dans son réseau")  
 if(getConsidereDansLeReseau(IP\_Machine\_B,trouver\_masque(Masque\_Machine\_B,Masque\_CIDR\_Machine\_B),IP\_Machine\_A,trouver\_masque(Masque\_Machine\_A,Masque\_CIDR\_Machine\_A))):  
 Element("result2").write("La machine B considère la A dans son réseau")  
 else:  
 Element("result2").write("La machine B ne considère pas la A dans son réseau")  
 except:  
 Element("result").write("Erreur dans les données entrées (masque ou IP erronnées?)")  
 Element("result2").write("")  
</py-script>

### « Login »

import bcrypt

import sqlite3

import webbrowser

con = sqlite3.connect("login.db")

cur = con.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS user(username, password)")

cur.execute("""INSERT INTO user(username, password) VALUES ('root', '$2a$09$9NZB5rQEVI4rrY1H1k/5zOZdxtI2jD3kSBkU1tknBbDEfBH7C/9IW')""")

def afficherMenu():

    print("Entrez votre nom d'utilisateur pour continuer...")

    usernameEntre=input()

    while(usernameEntre.replace(" ", "")==""):

        usernameEntre=input()

    print("Vous essayez de vous connecter en tant que ",usernameEntre)

    print("Veuillez entrer votre mot de passe...")

    cur.execute("select password from user where username like ?",(usernameEntre,))

    mdpVerifHash = cur.fetchone()

    mdpVerifHash = mdpVerifHash[0]

    mdpVerifHash = mdpVerifHash.encode('utf-8')

    utiliserMDP(mdpVerifHash)

def utiliserMDP(mdp):

    mdpEntre=input()

    mdpEntre=mdpEntre.encode('utf-8')

    if(bcrypt.checkpw(mdpEntre,mdp)):

        print("Mot de passe corest, renvoi vers l'application...")

        webbrowser.open('https://projetsysrso.web.app/Accueil.html')

    else:

        print("Mot de passe incorrect, veuillez réesayer...")

        utiliserMDP(mdp)

afficherMenu()